Path Planning For Manipulator Robots In Cluttered Environments

SAMIR LAHOUAR
Laboratoire de Mécanique des Solides
UMR: 6610 CNRS
Bd Pierre et Marie Curie BP 30179, 86962 Futuroscope chasseneuil Cedx
FRANCE

SAID ZEGHLOUL
Laboratoire de Mécanique des Solides
UMR: 6610 CNRS
Bd Pierre et Marie Curie BP 30179, 86962 Futuroscope chasseneuil Cedx
FRANCE

LOTFI ROMDHANE
Laboratoire de Génie Mécanique
Ecole Nationale d'Ingénieurs

Monastir 5019
TUNISIA

Abstract: - In this paper we propose a new path planning method for a robot manipulator in a cluttered static environment, 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 toward its goal. Then a width search mode becomes active when the robot gets close to an obstacle, and it 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.

Key-Words: - Path Planning – Manipulators – Lazy grid

1 Introduction

Path planning is an important issue in robotics. People recently are using randomized sampling based motion planners. Although these planners give remarkable results with many degrees of freedom (DOF), they are not complete by the way they use random landmarks to describe the configuration space. Examples of these methods are: Randomized path planner [1], probabilistic roadmap planners [2-6], Ariadne's Clew [7], Rapidly exploring Random Trees [8].

Since the introduction of the idea of using the configuration space to resolve path planning problems by Lozano-Pérez and Wesley in 1979 [9], almost all later works used configuration space called C space. In configuration space, the robot is reduced to a point representing its joint variables. Obstacles are transformed to a set of points that correspond to configurations where the robot is in collision with these obstacles. This set of points is called C space obstacles. C free space is the set of robot configurations that do not collide with any obstacle. Path planning consists of finding a path in the C free space from an initial point to a goal point.

The most difficult problem is how to represent the C free space which is equivalent to representing the C space obstacles.

Constructing explicit representation of C free space is very hard and needs a great running time especially if the number of primitives of robot and obstacles is big [10-12]. For that reason, sampling based approaches are proposed. There are different kinds of sampling based approaches: cell decomposition [13], grid based approaches [14-15], randomized planners. [1-8]

Probabilistic roadmap planners have been successful in many kinds of path planning problems especially in applications involving robots with many degrees of freedom. However they have a major problem in finding valid configurations in tight areas which makes them impractical in cluttered environments. Some methods where proposed to solve that problem, such as using the workspace medial axis[16]. In that method the medial axis of the workspace is generated and used in order to find configurations in tight regions.

In grid based approaches the C space is discretized to a sufficiently fine resolution. A collision detection method is used to decide whether a cell is in C free space or not, which creates a bitmap of C space. To find a path, a classical AI search technique is used.

Lengyel et al. [14] used a dynamic programming technique to find a path in 3 dimensional C space. However, in general this kind of approach is limited to lower dimensions since the number of resultant grid cells grows exponentially with the number of DOF and the resolution required to solve the problem.

Kondo planner [15] is based on the fact that it is possible to find a solution without visiting a large portion in the grid. Collision checking is delayed until a cell is needed to be checked. In that manner, the expensive processing step is avoided. A cost, f(c), is used in order to direct the search in the grid.

The cost f(c) is composed of two parts f(c)=g(c)+h(c) where g(c) is the standard cost-to-come and h(c) is a heuristic weighted sum of squares cost. The efficiency of the planner depends strongly on the heuristic used. The heuristic itself may be adapted for a kind of problems but not for another.

In our approach, we propose to compute only cells used to find the path and we use constraints in order to consider whether a cell is accessible or not, which enables us to use a bigger step in the grid. If we are far from obstacles we use only the distance to the goal in order to choose the cell to compute. If we are near an obstacle, we construct all the cells in order to find the best way to avoid this obstacle.

The planner we propose is based on the alternation of two searching modes. The first is a depth search mode active when the robot is far from obstacles, so it evolves toward its goal. The second is a width search mode when the robot is near an obstacle it permits to find the best way to avoid the obstacle.

2 Construction Of The C Space

A classic global method using a grid checks all its cells for collision before starting to search for a path. The number of the grid cells grows exponentially according to the number of DOFs of the robot. And in the same way, the time and the memory space required to compute and store the grid increases. While if the step of the grid is raised in order to reduce the number of cells, it is not certain to find a path without collision between two neighboring cells. For these reasons, we use the constraints proposed by [17] which in one hand makes the path between two neighboring cells in the C free safe even if the step is quite large, and in the other hand speeds up the collision checking process as the constraints computed in a cell are useful to check all the neighboring cells. The constraints calculated in a cell allow us to judge whether a path exists to a neighboring cell or not. Therefore, the constraints calculating process is equivalent to 3^N-1 times the collision checking process. As a cell has 3^N-1 neighbors. Where N is the number of DOF of the robot.

Fig. 1 shows a PUMA robot placed next to an obstacle. The constraint corresponding to that obstacle, as defined by [17], is written in that manner

$$\vec{\mathrm{V}}_{x_1 \in R_1 / R_0} . \vec{\mathrm{n}} \leq \xi (d - d_s) \qquad (1)$$

Where:

x_1 is the nearest point in the robot to the obstacle.

$\vec{\mathrm{V}}_{x_1 \in R_1 / R_0}$ is the velocity of the point x_1.

ξ is a positive factor.

d is the distance between the robot and the obstacle.

d_s is the security distance that the robot should not go beyond.

We consider $\mathrm{J}_{x_1}(q)$ the Jacobian matrix of the robot in point x_1 and Δq the configuration variation between two cells in the grid, we write the constraint as follow

$$\mathrm{J}_{x_1}(q) \Delta q . \vec{\mathrm{n}} \leq \xi (d - d_s) \qquad (2)$$

Which can be written as:

$$\mathrm{J}_{x_1}^{\mathrm{T}}(q) \vec{\mathrm{n}} . \Delta q \leq \xi (d - d_s) \qquad (3)$$

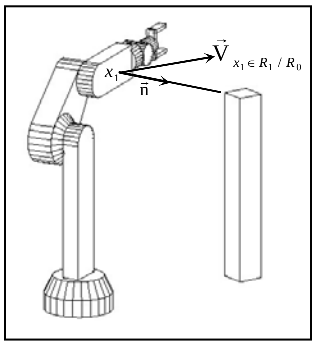

Fig. 1 Distance between a robot and an obstacle

In our approach we do not construct the entire grid, representing the C space, but we only construct the cells necessary to find the path to the goal position. If the robot environment is static, we can save the constructed cells to speed up the planning of other paths reaching other goals. Our planner uses two modes the first makes the robot evolve to its goal position if there is no obstacle that obstructs 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 we just store the cells near the obstacles which are sufficient to describe the C free space.

3 The Proposed Algorithm

In the following subsections we introduce some notions necessary for our approach.

3.1 The Cell Class

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

3.1.1 Pointer to the parent cell

As the grid is constructed while searching for the path from the initial configuration to the goal configuration, the first cell is the initial configuration it takes a NULL value. Then an already constructed cell is chosen according to the mode of the planner, the non-collision constraints are calculated for that cell and verified for the neighboring cells that have not been yet constructed. The cells verifying the constraints have a pointer to the cell used to construct them. If the goal cell is reached it just takes going back through the pointed cells to find the path between the goal and the initial cells.

3.1.2 A configuration defining a posture of the robot

Each cell corresponds to a point in the C space. If a cell configuration is written as $q_1 = \begin{bmatrix} q_1^1 & \ldots & q_N^1 \end{bmatrix}^T$ where N is the number of DOF of the robot. And let Δq be the step of the grid. The neighboring cells are defined as the configurations belonging to the following set:

$$\text{Vic}(q_1)=\left\{ q = \begin{bmatrix} q_1^1 + s_1\Delta q & \ldots & q_N^1 + s_N\Delta q \end{bmatrix}^T ; (s_1,\ldots,s_N) \in \{-1,0,1\}^N / (0,\ldots,0) \right\} \quad (4)$$

3.1.3 A distance to the goal

It represents the distance 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 the straight line in C space.

3.1.4 A boolean "collision" variable

It takes false if the cell verifies the constraints and true if it does not.

3.1.5 A boolean "computed" variable

Used by the planner in order to know whether the cell has already been used to search for the path or not.

3.1.6 A boolean "near an obstacle" variable

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.

3.2 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 $\text{h(Q)=t(Q)=}\varnothing$.

In order to handle the Queue Q, we use some operators that we will define next.

$\text{h}^+(\text{Q,c}_1)$ adds the cell c_1 to the head of Q.

$\text{t}^+(\text{Q,c}_1)$ adds c_1 to the tail of Q.

$\text{h}^-(\text{Q})$ removes the head cell from Q.

$\text{t}^-(\text{Q})$ removes the tail cell from Q.

3.3 The Stop Condition

We define the stop condition as the condition for which we judge that the goal position have been found. We write this condition as follows

$$\left\| q_{goal} - q \right\| < \Delta q \qquad (5)$$

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.

3.4 Algorithm

The algorithm outlined in fig. 2 begins by constructing the initial cell in step 1. It sets the parent pointer to NULL and evaluates the distance to goal. The algorithm uses a variable c representing the cell searched by the algorithm. $\aleph$ is the set of explored cells and $\aleph_1$ 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 verified. A cell is considered a collision if it does not verify constraints given by eq. (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 by step 17. This makes the planner perform a depth search since there is no obstacle bothering the robot. Whereas if the cell computed by step 15 is a collision, all non-collision cells in the vicinity of c and close to collision cells are placed in the tail of the queue Q by step 22. Which makes the planner perform a depth search till the obstacle is bypassed.

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.

```
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.      $\aleph_1 = vic(c) \backslash \aleph$
8.      For each cell $c_2 \in \aleph_1$ do
9.          Set $c_2$.parent = c
10.         Evaluate $c_2$.distance_to_goal
11.         Verify the non-collision constraints and determine $c_2$.collision
12.         Set $c_2$.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 $c_3$.collision=false then
17.         $h^+(Q,c_3)$
18.     Else ($c_3$.collision=true)
19.         For each $c_2 \in vic(c)$ such as $c_2$.collision=true do
20.             For each $c_3 \in vic(c_2) \cap \aleph$ set $c_3$.near_an_obstacle=true
21.         End for
22.         For each $c_2 \in vic(c) \backslash Q$ such as $c_2$.Near_an_obstacle = true and $c_2$.collision=false and $c_2$.computed=false do $t^+(Q,c_2)$
23.         For each $c_2 \in Q$ such as $vic(c_2) \subset \aleph$ remove $c_2$ from the Queue Q and set $c_2$.computed=true
24.     End if
25.     $c=h(Q)$
26.     $h^-(Q)$
27. End while
```

Fig. 2 Pseudo-code of the method

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 throw 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.

4 2D Example

In order to illustrate the proposed algorithm we give a 2D example, of a 2R robot (Fig. 3) evolving among point obstacles. We make simulations using three point obstacles defined by table 1. The start configuration is $\theta_s = [-20° \quad 30°]^T$ and the goal configuration is $\theta_g = [40° \quad -40°]^T$. Fig. 4 shows the robot in its starting and goal positions and the three point obstacles. We set the lengths of the arms of the robot $l_1 = l_2 = 10$.

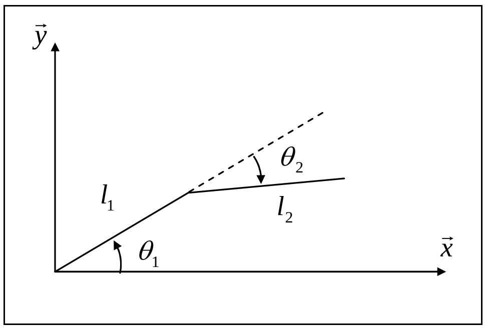

Fig.3 2R Robot

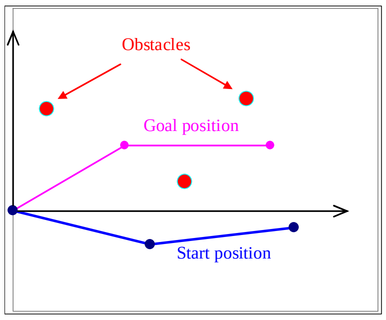

Fig.4 Path planning consists of moving the robot from the start position to the goal position without colliding with obstacles

Fig. 5 shows the C space of the robot, the dark regions correspond to C space obstacle. The construction order of cells is shown in fig. 6. The algorithm evolves toward 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.

Obstacle	O_1	O_2	O_3
x	16	2,31	11,8
y	11	10	2,8

Table 1.

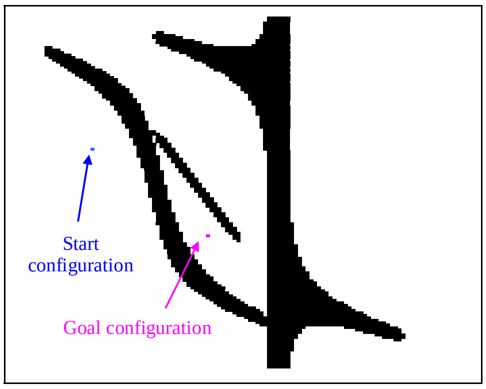

Fig. 5 Configuration Space

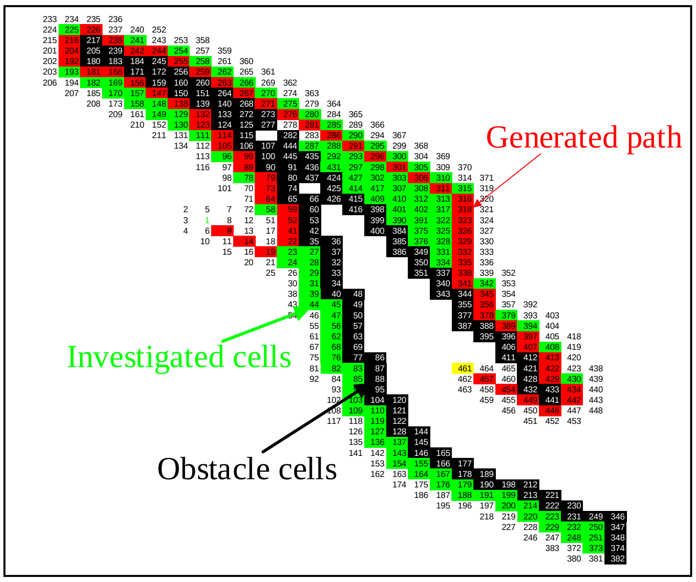

Fig. 6 Cell generation order

The algorithm gives the best way to go around the C obstacle. The result of the simulation is shown in fig.7. Moreover, out of 4891 cells, only 461 cells were computed which represents less than 10% of the whole workspace.

5 Simulation and Results

Simulation has been performed on a robotic-oriented-Software named SMAR [18]. This software is composed of two units: a modeling unit and a simulation unit. The Modeling unit is used to generate a model of the robot in its environment. The simulation unit is used to simulate the motion of the robot in its environment. It contains a minimal distance feature we used to implement our algorithm.

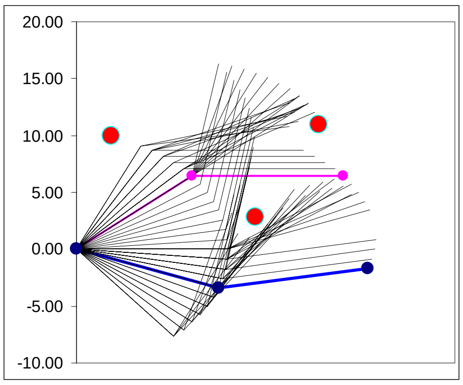

Fig. 7 Results

Fig. 8 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. Our 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 Figure 5, show 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 represent less than 2% of the whole workspace.

6 Conclusion

In this paper we presented a new method of path planning based on lazy grid methods and searching for path without using a heuristic function. This method reduces the gap between classic grid methods where all the grid cells must be computed before searching for a path, and lazy grid methods where the grid is computed while searching for the path. The proposed planner is very general and is guaranteed to find a path if one exists a given resolution. However, this algorithm depends on the resolution of the grid. The higher this resolution is, the closer the robot can squeeze between obstacles.

This method reduces the number of computed cells and gives the best direction to go around a C obstacle. It can be combined with quasi-random methods and replace the A* searching module. Where quasi-random sampling of the C space appears to offer performance improvements in path planning [19].

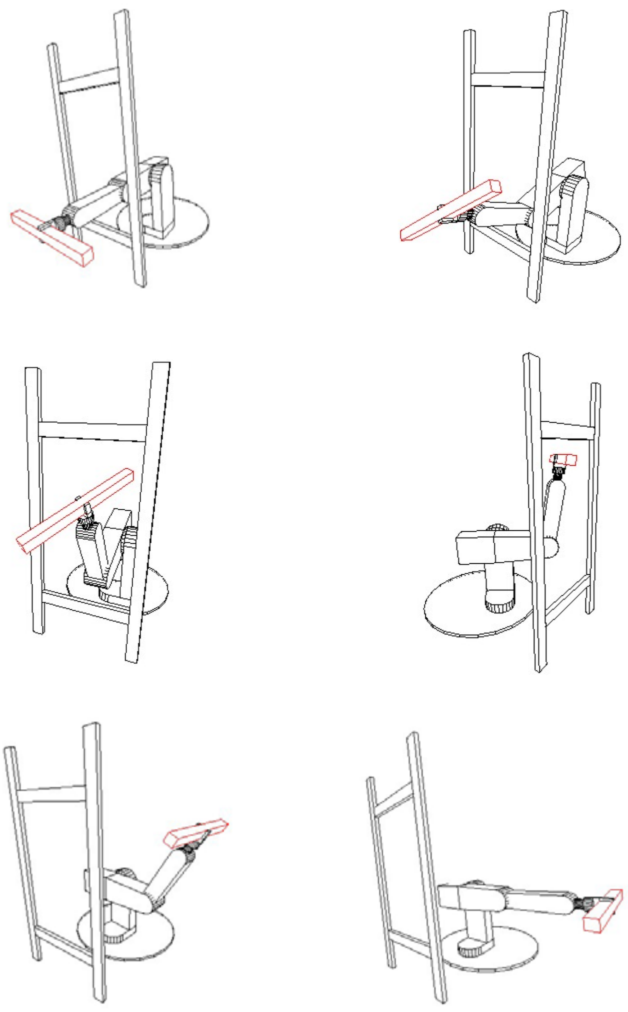

Fig. 8 Results

References:

[1] J.Barraquand and J. C. Latombe, A Monte-Carlo algorithm for path planning with many degrees of freedom, *International Conference on Robotics and Automation*, 1990, pp. 1712-1717.

[2] L.E. Kavraki, P. Svestka, J-C. Latombe and M.H. Overmars, Probabilistic roadmaps for path planning in high dimensional configuration spaces, *IEEE Transacrions on Robotics and Automation*, Vol.12, No.4, 1996, pp. 566-580.

[3] T Siméon J-P. Laumond and C. Nissoux, Visibility-based probalistic roadmaps for motion planning, *Advanced Robotics Journal*, Vol.14, No.6, 2000, pp. 477-493.

[4] S. A. Wilmarth, N. M. Amato and P. F. Stiller, MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space, *International Conference on Robotics and Automation*, 1999, pp. 1024-1031.

[5] R. Bohlin and L. E. Kavraki, Path Planning Using Lazy PRM, *International Conference on Robotics and Automation*, 2000, pp. 521-528.

[6] P. Leven and S. Hutchinson, A framework for real-time path planning in changing environements, *The International Journal of Robotics Research*, Vol.21, No.12, 2002, pp. 999-1030.

[7] E. Mazer, J.M. Ahuactzin, and P. Bessière, The Ariadne's clew algorithm, *Journal of Artificial Intelligence Research*. Vol.9, 1998, pp. 295-316.

[8] S. M. LaValle and J. J. Kuffner, Randomized Kinodynamic Planning, *The International Journal of Robotics Research*, Vol.20, No.5, 2001, pp. 378-400.

[9] T. Lozano-Pérez and M. A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Communications of the ACM*, Vol.22, No.10, 1979, pp. 560-570.

[10] J. T. Schwartz and M. Sharir, On the piano mover's problem: II. General techniques for computing topological properties of algebraic manifolds, *Advances in Applied Mathemathics*, Vol.4, 1983, pp. 298-351.

[11] J. F. Canny, *The complexity of Robot Motion Planning*, MIT Press, 1988.

[12] S. Basu, R. Pollack and M. F. Roy, *Algorithms in Real Algebraic Geometry*, Springer-Verlag, 2003.

[13] B. Paden, A. Mess and M. Fisher, Path planning using a Jacobian-based free space generation algorithm, *International Conference on Robotics and Automation*, 1989, pp. 1732-1737.

[14] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg, Real-time robot motion planning using rasterizing computer graphics hardware, *Computer Graphics*, Vol.24, No.4, 1990, pp. 327-335.

[15] K. Kondo, Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration, *International Conference on Robotics and Automation*, 1991, pp. 267-277.

[16] C. Holleman and L. E. Kavraki, A framework for using the workspace medial axis in PRM planners, *International Conference on Robotics and Automation*, 2000, pp. 1408-1413.

[17] B.Faverjon and P. Touranssoud. A local based approach for path planning of manipulators with a high number of degrees of freedom. *International Conference on Robotics and Automation*, 1987, pp. 1152-1159.

[18] S. ZEGHLOUL, B. BLANCHARD and M. AYRAULT, SMAR: A Robot Modeling and Simulation System. *Robotica*, Vol.15, No.1, 1997, pp. 63-73.

[19] M.S. Branicky, S.M. LaValle, K. Olson and L. Yang, Quasi-randomized path planning, *International Conference on Robotics and Automation*, 2001, pp. 1481-1487.