ISSN 0005–1144
ATKAAF 43(3–4), 131–137 (2002)

Mladen Crneković, Davor Zorc, Dubravko Majetić

Robot Motion Planning by Limited Space Method

UDK 62-526
004.896
IFAC IA 5.9.3;2.5

Preliminary communication

Robot motion planning in multidimensional space is very time-consuming and requires a big model; therefore, it is not very suitable for a real-time purpose. Limited space method (LSM) used here works with 3D real physical space (two translations and one rotation) and finds out a logical path (in the sense of the human solution faced with the same problem), rather than the optimal path (in the mathematical sense). Its main advantages are small model and short solution time. Although LSM is not as universal as the C-space, it has good potentiality for engineering applications.

Key words: collision detection, motion planning, obstacle avoidance, limited space method

1 INTRODUCTION

If you skim over articles dealing with robotics from the last 10 years, you will see that the most often explored topic is – how to make a robot behave intelligently.

Of course, there is no simple and wide accepted definition of intelligence, but almost everyone will agree that the basic prerequisite condition for the robot intelligent behavior is its ability to move in the environment and reach the goal without colliding with any obstacle.

In fact, there are two separate problems: mobile robot movement and robotic arm movement, but there are many similarities between the solution methods. This article is focused on the robotic arm movement in the environment with obstacles. The final solution of such a problem is path calculation that is safe for the robot, i.e. the robot can avoid collision with any obstacle.

Of course, the exact mathematical solution of this problem is well known. The method is called configuration space (C-space) and leads to a combinatory problem where we must search the solution among a few million elements in multi dimensional space [4]. This method produces a big model, it is time-consuming and gives the best result. But, for engineering and real time purposes, it can hardly be applied. Therefore, some simplifications are necessary to avoid the method limitations. These simplifications will produce a pseudo-optimal result.

In the real world we do not want the best result (optimal, in the mathematical sense), because it is too expensive. We will be satisfied with the result that is close to the best, but with the acceptable solution time. So, let us go back from the C-space to the physical (»real«) space. Here are some ideas from previous articles.

Paper [1] starts by the fact that two objects in a space can be in 6 different positions relative to each other. Apart from the condition that the objects must be convex, there are no limitations in the number of plains the objects are composed of. Typical bodies are investigated (cubes, rectangular boxes, cylinders, cones) that can build up a robot geometry model. The proposed algorithm is fast, and after initialization it gives the Euclidean distance of the objects often in 3 to 4 ms, but it does not calculate any path.

Paper [2] presents an algorithm which replaces objects with a set of spheres with different size and distribution. It can simulate even concave objects, but generates pseudo-optimal solutions for collision detection. Solution time for the collision detection is from 1.1 to 2.8 seconds (up to 10.5 s for »normal hard« model). As in reference [1], the algorithm does not calculate a robot path. To reduce the solution time, the authors have decided to use parallel algorithm and up to 11 processors.

Instead of using planes, paper [3] defines possible collision points (two spheres per robot link), therefore simplifies the problem. The concept exploits the recursive forward kinematic structure and is not based on the configuration space representation. Although 10-axis manipulator is modeled, there is no information about solution times.

In [5] the Puma 560 robot has been described by 509 spheres, while obstacles are represented as a weighted voxel map (value of any voxel indicates its distance to the nearest obstacle). For a 2-meter cube working space and voxel resolution of 2 cm, 1 Mbyte of memory is required for the voxel map. Its generation required around 15 seconds. After that, collision detection routine was executed within 10 ms (486 PC, 66 MHz). For a given example the path-planning time was 6 seconds. Since n-dimensional space transforms into n 2-dimensional subspaces, the time for the path searching is reduced.

Paper [7] introduced a similar idea called distance map (d-map). Workspace is represented as a 3D-grid. Each grid cell has its distance value (a big value for the free region, and zero value for the obstacle region). After that, the method is a mix of the C-space and the potential field method. To avoid a big model and searching in the 5-dimensional space, Kohonen's self-organizing map for the C-space reorganization has been used. The total execution time rises up to several tens of seconds. Before that, C-space off-line construction takes a few hours. Although it seems a long time, the problem that was solved was very complex.

In paper [8] the author starts with a similar idea as in the presented limited space method. He restricts the free space concerning path planning and avoids unnecessary collision detection. This restriction is in the C-space, rather than in the physical space. Multiple search strategies are executed (heuristic functions with different coefficients), and a more promising one is selected. Configuration space contains 2^{42} cells (approximately 4 thousand billion) and is organized as a 64-branch tree in 7 levels. Unfortunately, no execution time has been reported, but according to the model size, the execution time will be huge.

Paper [9] classified the path planning methods with a short description of each of them. In [10], the method of equidistant path has been presented in details. Although the presented method could be used for an articulated arm fixed in the work cell, the method is more suitable for mobile robots.

In papers [11], [12] and [13] the authors give us a good representation of the potential field method applied on both mobile robots and articulated arms. Although the method has a good mathematical background, it rarely appears in papers published in the last several years. The main reason for that could be generation of the local minima in the potential field. Generally, these local minima could be avoided, but then the whole method looses its simplification.

The purpose of this work is not to find an optimal (in the most case the shortest) path, but rather a »logical« path (in the sense of a human solution faced with the same problem). To avoid a big model and a long execution time, only searching in the »real« space has been performed (i.e. physical space). Robot orientation on the path is determined using a flexible heuristic algorithm. Depending on the desired space resolution, the solution is obtained in a few seconds, which is slightly more than a human reaction faced with the same problem, but significantly less than in the C-space. Before application on a real robot with 5 DOF (Mitsubishi RM 501), the method is investigated in the plane on a robot with 3 DOF (x, y, φ).

The main advantage of the proposed method is that it reduces the model from 3D space (two translation and one rotation) to 2D space by limiting the robot orientation. Therefore, the required memory and path searching time could be significantly reduced. Of course, memory and time reduction has its cost. In an extreme situation (such as a sudden great change in orientation) the algorithm could fail, and then the second iteration is necessary. According to the previous result, the next iteration could be less orientation-limited, which results in the longer execution time.

Motion planning for the standard robot task (such as pick and place) could be solved in a few seconds, that is the time for the physical execution of the task.

The most often collision with the environment will happen by the upper robot arm. That is the reason why the method takes into account only the tip of the tool, instead of the whole robot's body.

Limited space method (LSM) consists of several parts: collision detection, heuristic orientation algorithm and space searching.

2 COLLISION DETECTION

Essential part of the LSM is collision detection between two bodies. In fact there are two interesting questions: 1) for two given bodies, whether they intersect or not, and 2) what is the minimal distance for given two bodies?

Although the distance between two bodies is very interesting and useful information, its calculation is not simple and takes a lot of computer time. Therefore, we will be satisfied only with the answer to the first question.

Giving a set of points for body $\mathbf{Q} = \{(x_i, y_i) \in \mathbf{R}^2, i = 1, N_Q\}$ and $\mathbf{W} = \{(x_i, y_i) \in \mathbf{R}^2, i = 1, N_W\}$, a simple mathematical calculation can answer the question: whether they intersect or not. The only condition on the set points is that they form an object that

has no concave shapes. According to figure 1 this is a well known problem of linear separability between two sets of points.

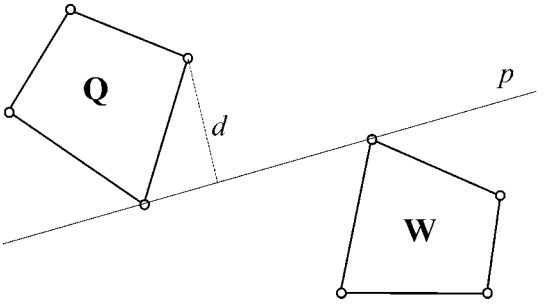

Fig. 1 Linear separability of two bodies in the plane

If two sets, **Q** and **W**, are linearly separable (i.e. are not in collision) then exists a line p, so that all points of set **Q** lie on one side, and all points of set **W** lie on the opposite side of the line p. But, if two sets are linearly separable, there exist an infinite number of lines that satisfy the previous condition. Therefore, the additional condition is that the line p contains minimum one point from the set **Q** and one point from the set **W**. Now, it is possible to design only $N_Q \times N_W$ lines between two sets. When we choose a point (x_Q, y_Q) from the set **Q**, and a point (x_W, y_W) from the set **W**, then the line between them can be expressed by the following equation:

$$\begin{aligned} &Ax + By + C = 0 \\ &A = y_W - y_Q \\ &B = x_Q - x_W \\ &C = -(Ax_Q + By_Q). \end{aligned} \tag{1}$$

A distance d between any point (x_i, y_i) and line (1) is:

$$d_i = \frac{1}{\sqrt{A^2 + B^2}}(Ax_i + By_i + C). \tag{2}$$

As we are intersted only in the sign of d_i, equation (2) can be simplified as:

$$d_i = \text{sign}\,(Ax_i + By_i + C). \tag{3}$$

We also need an algorithm that determines on which side, relative to the line p, a body **G** lies. The required algorithm is »BodySide«. If all body set points lie on one side relative to the line p, the algorithm returns 1, if all body set points lie on the other side, the algorithm returns –1. If the body points lie on different sides of line p, the algorithm returns 0.

Algorithm BodySide (Q, W)
. **if** $d_i[\mathbf{G}(x_1, y_1)] > 0$ **then**
BodySide = 1 **else** BodySide = –1
. **for** every $G(x_i, y_i)$; $i = 1, N_G$
. **if** $d_i[G(x_i, y_i)] < 0$ **then**
{BodySide = 0; **break** algorithm}
. **next** i

Finally, the complete algorithm for collision detection is as follows:

Algorithm Collision (Q, W)
. suppose collision is True
. **for** every $Q(x_i, y_i)$; $i = 1, N_Q$
. **for** every $W(x_j, y_j)$; $j = 1, N_W$
calculate A, B, C from equation (1)
calculate SideQ = **BodySide** (Q) algorithm
calculate SideW = **BodySide** (W) algorithm
if (SideQ × SideW) ≥ 0 **then** continue
else {collision is False; **break** algorithm}
. **next** j
. **next** i

If bodies **Q** and **W** do not intersect, the collision algorithm can stop when this condition is detected. Only in the case when the bodies are in collision, the inner loop of the collision algorithm must be executed $N_Q \times N_W$ times. It is interesting to notice that $d = 0$ from equation (3) means: no collision.

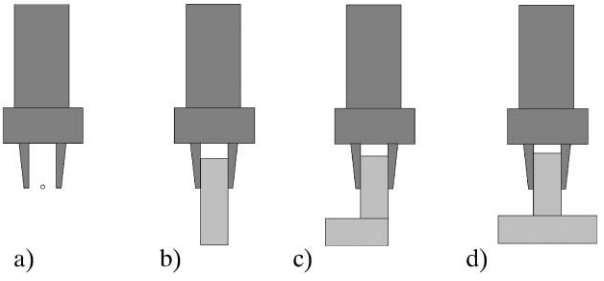

Fig. 2 Hand geometry (a) and different objects (b, c, d)

This method of collision detection is used to detect collision between any obstacle and a robotic hand (including an object in hand). To describe the robotic hand, it is necessary to use more than one body. Figure 2 a) shows a possible hand geometry which consists of four bodies. Each body is a set of four points. The object in the robotic hand can also be described with a set of bodies (figure 2 b, c, d). In that case any shape of a body (which consists of lines and planes) can be designed even if it has concave parts.

An analog concept of linear separability may be defined in 3D. The only difference is that the line p becomes the plane p.

3 LIMITED SPACE METHOD SEARCH

Although the problem is described is three dimension space (x, y, φ), it will be solved in two dimensions (x, y). The third dimension, hand orientation φ, will be limited by the robotic hand position. That is why the method is called »limited space«.

Physical space has dimensions $L_W \times L_H$ and is divided into cells. Each cell has a dimension D_{xy}. Also, the robot orientation of 360 is divided in step D_φ. So, the total number of cells in 3D space is $N_{cell} = L_W/D_{xy} \times L_H/D_{xy} \times 360\ /D_\varphi$. For $L_W = 600$, $L_H = 300$, $D_{xy} = 10$ and $D_\varphi = 10$, N_{cell} is 64 800. But in the »limited space« (2D space) the total number of cells is much less, only 1800.

Each cell has its state: -1 = not defined, 0 = free of collision or 1 = occupied (in collision). This state is valid only for a determined orientation. The starting robot position has a label »START cell«, and the goal has a label »GOAL cell«. It is necessary to have an algorithm for searching from START to GOAL cell. From the infinite number of solutions, a searching process must select a path which is the shortest, or close to the shortest possible one.

For this purpose the Dijkstra graph search algorithm will be a good base. Of course, it needs some modification and adaptation to a specific situation. First of all, the graph is a grid; therefore the graph structure is well known. This adaptation leads to something known as a grid search algorithm. During the graph (grid) searching process, two important functions have been calculated. For each cell they are defined as:

$$\begin{aligned} L &= L^* + \Delta l \\ L_P &= L_P^* + \Delta l + \beta\Delta\varphi + \tau g. \end{aligned} \tag{4}$$

L is the function that will be used for the robot orientation definition. It has the meaning of the shortest path from the start position to the current cell. L^* is L in the previously connected cell with distance Δl. Because the graph is a grid, Δl can only be D_{xy} or $\sqrt{2}D_{xy}$.

L_P is the cost function that will be minimized in the searching process. It has the meaning of the pseudo-length, and can consist of several cost parts. First, it is the Euclidian distance Δl from the previously connected cell (index $*$). To include the robot orientation in the optimization process, each cell can be weighed by changing the robot orientation $\Delta\varphi$ from the previously connected cell. The weight factor β determines the relation between translatory and rotational motion.

Because the collision detection algorithm does not calculate the distance from obstacles, the robot will pass very close to them (it can even touch them). To avoid touching and to keep the robot away from obstacles, additional cost part g in the pseudo length function has been added. Variable g determines the number of potentially occupied cells that surround the current cell (i.e. where collision will occur).

The weight factor τ determines the relation between the translatory motion and g number. For $\beta = 0$ and $\tau = 0$, $L_P = L$ that means the pure distance function has been optimized.

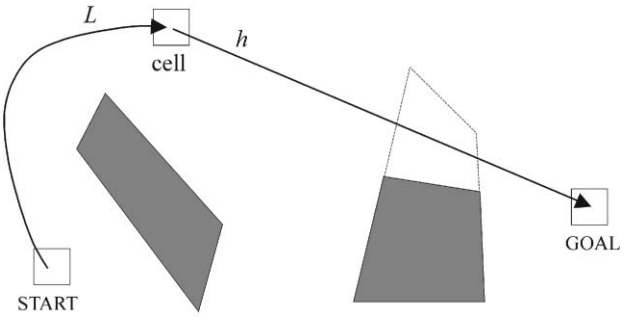

Fig. 3 Start-cell distance L and cell-goal heuristic distance h

The second modification of the Dijkstra graph searching algorithm is known as the grass fire searching method. It means that the next possible path point in the searching process is not chosen only depending on the START – cell pseudo-distance L_P, but also depending on a cell – GOAL distance h. Of course, in the searching process distance h is not known (because the complete path is not known), therefore it is calculated as a heuristic function, figure 3. The most common search function is the Euclidian distance between a cell and the GOAL. Therefore, the total decision function L_D for each cell in the searching process is:

$$L_D = L_P + \alpha h, \tag{5}$$

where α is a weight factor for heuristic function h. As L_P and h are of the same dimension (distance or pseudo-distance) the most logical value for α is 1. If α is greater, the designed path is often further away from the shortest one, but the solution converges faster. For α reasonable compromise between the closest path and a short solution time, it is good to select $1 < \alpha < 2$.

Space building and the searching process are integrated in the same algorithm; therefore robot collision detection is checked only for a part of the working space. This approach speeds up the searching process for one degree of order. In addition, a separate array of cells is built. This array consists of all cells which have the state »free of collision«.

In the phase of searching the minimum cost function (L_P), only elements in this array are investigated instead of all the space. This speeds up the searching process significantly, but it requires some additional memory.

Now, let us see what exactly the term »limited space« means. In any cell there are infinite number of orientations. But if the robot orientation changes in steps D_φ, then the number of possible orientations is reduced to 360 $/D_\varphi$. For $D_\varphi = 10{,}36$ different rotations are possible in each cell. For some robot orientations, a cell is free of collision, for some it is not. The question is for which orientation in a particular cell the collision detection will be calculated.

Even if we do not known the robot orientation along the path (in fact, even the path is not known yet), it is reasonable to suppose that the robot orientation will be a smooth function from the START to the GOAL position. In the first approximation, the robot orientation from the START to the GOAL position will change linearly. Of course, this is only a presumption. In reality the robot orientation function can be different from the linear one and can be adapted to a specific situation, but in a »limited« way. Figure 4 illustrates an idea of changing and adapting the linear orientation.

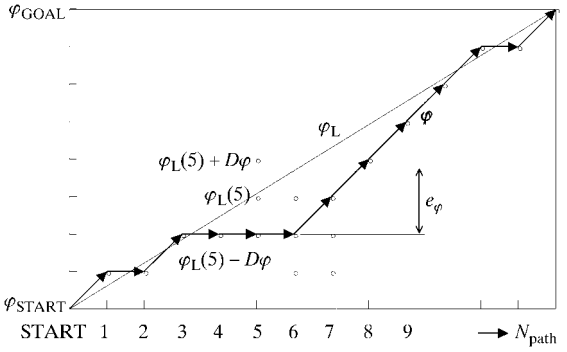

Fig. 4 Robot orientation modification

If the robot orientation on the START position is φ_s and on the GOAL position φ_G, then the linear orientation φ_L on the fraction f of the path will be

$$\varphi_L = \varphi_s + f(\varphi_G - \varphi_s) \tag{6}$$

where $f = 0...1$ is a part of the path length L_{path}. Because this part is not known yet, we estimate it for each cell as

$$f = \frac{L}{L+h}. \tag{7}$$

Distance L is well known, but if the rest of the path is not a straight line (from the cell to the GOAL), the value of the heuristic function f is estimated too low. Therefore, the part length f will often be overestimated.

Because the robot orientation φ can change in steps of D_φ, the linear orientation φ_L is rounded to the first nearest value that can be divided by D_φ. If the robot can reach this position by φ_L orientation, then φ_L is the solution. There are really rare occasions when this will happen along the whole path. Very often the robot must modify its orientation from φ_L. This modification can be only close to the robot orientation φ of the previous cell φ^*, and that is the reason why this method is called »limited space«. »Close to the previous cell φ^*« means $\varphi^* \pm D_\varphi$, depending on the sign of the robot orientation error $e_\varphi = \varphi_L - \varphi$. If in these two additional orientations the robot is in collision, this cell is definitely occupied (in collision). Of course the orientation $\varphi^* \pm 2D_\varphi$ or $\varphi^* - iD_\varphi$ that is free of collision is also possible, but this solution will not follow the principle of the orientation »smoothness«.

Whenever φ differs from φ_L, there is a tendency to come back to φ_L, but only in D_φ steps. This forces the robot orientation to approach the GOAL orientation.

The difference in φ from cell to cell can also be taken into account in pseudo-length calculation L_P, equation (4).

Of course, there are problems that cannot be solved using this method as they need more extensive methods such as C-space. But LSM is oriented to engineering applications where it is not so important to have an optimal path. Instead of that, the solution may be near to the optimal one (it has to be logical in the human sense) but the solution time must be acceptable.

4 EXAMPLES

Let us investigate LSM on some examples. The space dimension is 600×300 units and the LSM algorithm runs with $\alpha = 1.5$, $\beta = 1$, $\tau = 10$, $D_{xy} = 5$ and $D_\varphi = 10$, if different conditions are not mentioned.

Figure 5 shows a simple case where the robot must move from one side of the obstacle to the other. Along the path, the robot orientation is unchanged. There are 55 points on the path having the length of 348 units. Despite the fact that the method does not know the distance between the robot and the obstacle, the robot passes not closer than D_{xy} from the obstacle. The solution time is obtained in one second, which is an acceptable result.

Figure 6 shows a similar robot task, but in the goal position the robot must approach a body to hold it. During the motion, the total change in the robot orientation is 90 . Although we expect that

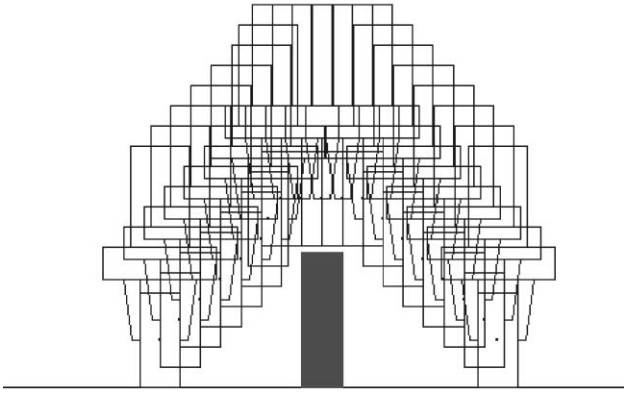

Fig. 5 Simple case of the obstacle avoidance

the robot plans to approach the body from the upper side, it did not happen. That is because the method optimized the pseudo-length (321 units) of the path, and not the robot approach.

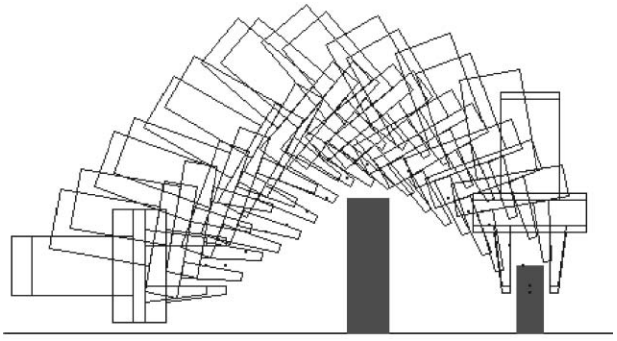

Fig. 6 Approaching and holding a body

It is often the case that the robot must draw out some object and put it on a working place. Figure 7 shows the planned motion for this kind of task. By this example we show the ability of the method to modify the robot orientation in order to draw out long objects that it holds. Finally, it changes its orientation to its goal value. The path consists of 58 points and has a length of 314 units. It is interesting to notice at this point that the opposite path planning (from GOAL to START) will fail. It happens when a sudden big change in the robot rotation is necessary, but there is no place in the »limited« space. The method is not yet flexible enough for solving this kind of a problem. In some situations the solution is simple: when the method fails, the START and the GOAL position can exchange places (instead of »draw in«, try »draw out«). This exchange can produce a solution because solutions are in general not the same in both directions (from START to GOAL and from GOAL to START). In one direction the space may be too »limited«, while in the opposite one it is not.

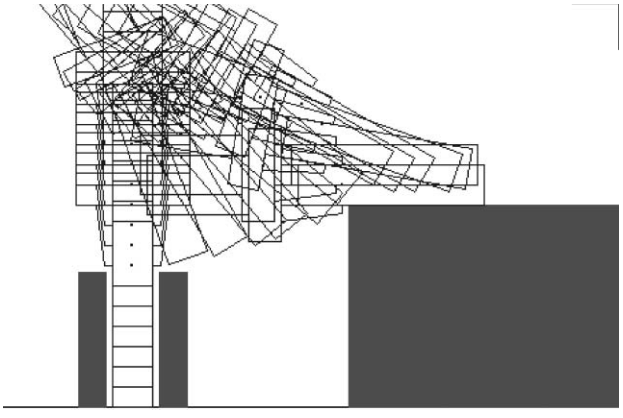

Fig. 7 Drawing out a body

Figure 8 is an example of a long path. It consists of 102 points, with the total length of 577 units. The robot orientation change from the start to the goal point is 180 and it follows the principle of a continuous movement.

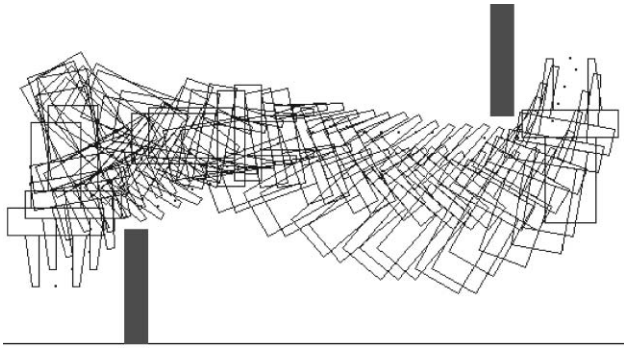

Fig. 8 Long path planning

The most difficult case is when the robot must insert an object through a tight pass, and rotate it at the same time. Figure 9 shows this case. If $D_{xy} = 11$, the method fails because the resolution is too rough. With a better resolution the path length is 458 units and as D_{xy} decreases, the changes in the

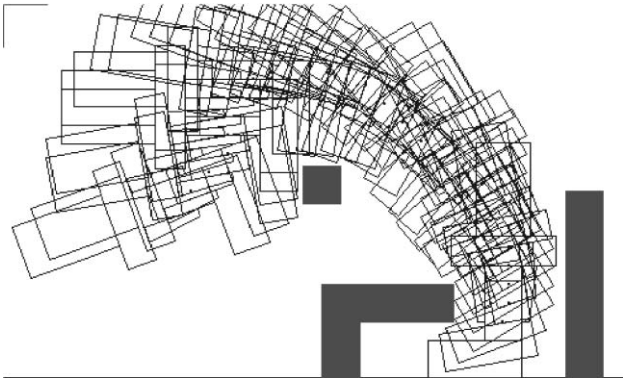

Fig. 9 Inserting through a tight pass

path construction are slight. But the solution time increases rapidly from 2.2 seconds ($D_{xy} = 10$) to more than 20 seconds for $D_{xy} = 3$.

All examples were executed on Duron 700 MHz processor, AGP video card and the algorithm is coded using Delphi language.

5 CONCLUSION

Although it is not a universal solution and it cannot solve all problems, LSM shows a good potentiality for engineering applications. It avoids the use of a big model and a long solution time by searching for a logical path. Tasks like picking and placing, approaching the object and inserting it in a hole can be solved in no more than a few seconds.

Further investigation is necessary in order to improve the robot orientation planning. Instead of planning only the robot linear orientation planning, several different scenarios could be planned. Even in the linear orientation planning, some improvements are possible. First, if the function L is replaced with the cost function L_P in equation (7), the final result will change only a little. This is because the heuri-stic function h is not known, and very often it is es-timated too low. On the other hand, L_P is often greater than L, therefore L_P compensates h. The second improvement in linear orientation planning is to make the »limited space« wider, and to permit ranges $\varphi^* \pm D_\varphi$ and $\varphi^* \pm 2D_\varphi$ for adaptation.

To speed up the process, the use of parallel processing is possible.

After additional investigation it is planned to expand LSM to the 3D linear grid (3 translations and two rotations) and to try to apply it for the motion planning of the Mitsubishi RM501 robot.

REFERENCES

[1] M. C. Lin, J. F. Canny, **A Fast Algorithm for Incremental Distance Calculation**. Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Vol. 2, pp. 1008–1014, 1991, Sacramento, California.

[2] M. Perez-Francisco, A. P. del Pobil, B. Martinez, **Fast Collision Detection for Realistic Multiple Moving Robots.** Proceedings of the 8th International Conference on Advanced Robotics, pp. 187–192, 1997, Monterey, California.

[3] M. Schlemmer, G. Gruebel, **A Distance Function and its Gradient for Manipulator On-Line Obstacle Detection and Avoidance**. Proceedings of the 8th International Conference on Advanced Robotics, pp. 427–432, 1997, Monterey, California.

[4] T. Lozano-Perez, **Spattial Planning: A Configuration Space Approach.** IEEE Transaction on Computers, Vol. C-32, No 2, pp. 108–119, February 1983.

[5] M. Greenspan, N. Burtnyk, **Obstacle Count Independent Real-Time Collision Avoidance**. Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Vol. 2, pp. 1073–1080, 1996, Minneapolis, Minnesota.

[6] R. Heine, T. Schnare, **Kollisionsfreie Bahnplanung für Roboter**. Robotersysteme 7, pp. 17–22, 1991.

[7] E. Ralli, G. Hirzinger, **A Global and Resolution Complete Path Planner for up to 6DOF Robot Manipulator**. Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Vol. 2, pp. 3295–3302, 1996, Minneapolis, Minnesota

[8] K. Kondo, **Motion Planning with Six Degrees of Freedom by Multistrategic Bidirectional Heuristic Free-Space Enumeration**. IEEE Transaction on Robotics and Automation, Vol. 7, No 3, pp. 267–277, 1991.

[9] M. Crneković, **Autonomous Mobile Robot Path Planning – Problems and Methods**. International Symposium: Flexible Automation, Strbske Pleso, pp. 30–31, 1991.

[10] M. Crneković, B. Novaković, D. Majetić, **Mobile Robot Path Planning in 2D Using Network of Equidistant Path**. Journal of Computing and Information Technology, Vol. 7, No. 2, June 1999.

[11] A. Graham, R. Buckingham, **Real-time Collision Avoidance of Manipulators with Multiple Redundancy**. Mechatronics, Vol. 3, No. 1, pp. 89–106, 1993.

[12] H. Noborio, S. Wazumi, S. Arimoto, **An Implicit Approach for a Mobile Robot Running on a Force Field Without Generation of Local Minima**. 11th IFAC Congress, Vol. 9, pp. 85–90, Tallin, 1990.

[13] L. Singh, H. Stephanou, J. Wen, **Real-time Motion Control eith Circulatory Fields**. IEEE Conference, pp. 2737–2747, Minnesota, 1996.

This research is supported by the Croatian Ministry of Science and Technology, Project No. 120008/1996.

Planiranje gibanja robota metodom ograničenog prostora. Planiranje gibanja robota u višedimenzijskom prostoru zahtijeva veliki model i traje predugo, te stoga nije pogodno za procese u realnom vremenu. Metoda ograničenog prostora (LSM), korištena u ovom radu, umjesto da traži optimalnu putanju gibanja (u matematičkom smislu), istražuje 3D realni fizički prostor (dvije translacije i jedna rotacija) i pronalazi logičnu putanju robota (logičnu u smislu čovjekovog rješenja istog problema). Njezina osnovna prednost je mali model i brzo rješenje. Iako metoda ograničenog prostora nije sveobuhvatna kao npr. metoda konfiguracijskog prostora, ona pokazuje dobru primjenjivost za inženjerske potrebe.

Ključne riječi: detekcija sudara, planiranje gibanja, izbjegavanje prepreka, metoda ograničenog prostora

AUTHORS' ADDRESSES:

Prof. Mladen Crneković, Ph. D.
Assoc. prof. Davor Zorc, Ph. D.
Assoc. prof. Dubravko Majetić, Ph. D.
Faculty of Mechanical Engineering and Naval Arhitecture
Ivana Lučića 1, 10000 Zagreb, CROATIA

Received: 2002–07–12