'94 KACC (1994. 10. 17 ~ 20)

Real-time Collision-free Path Planning for Robot Manipulator

Koichi Hamada and Yoichi Hori
Department of Electrical Engineering, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan

Abstract – **This paper presents a real-time calculation method to generate the trajectory of robot manipulator for the purpose of avoiding collision. In order to model 3-D workspace, we use octree which has been used for fast collision detection. The levels of octree are used as the cost function to represent the distance between the manipulator and the obstacles. This criterion is not exact, but, due to this, we can obtain the approximate feasible trajectory extremely quickly. We will show the effectiveness of our method with some simulation examples. For example, the proposed method can solve a problem within 1 second on Intel 80486 processor running at 33 MHz. It has taken more than half an hour with one of the previously proposed methods.**

1 Introduction

With the extended use of moving robots in manufacturing and nuclear industries, the problem of collision-free path planning has become more important. The collision-free path for manipulator should be planned automatically in a real-time manner to improve efficiency of work. In this paper, we propose a real-time calculation method to generate the collision-free trajectory of robot manipulator.

In the previous papers, many methods to plan the trajectories for robot manipulator have been proposed. It is known well that the methods using the Configuration Space [8] are effective in collision avoidance. As large computer memory is necessary for the Configuration Space, the way to reduce the amount of data is also studied.

Another method, the potential field approach [3], [6], [7] uses potential functions for obstacle avoidance. The obstacles are assumed to have electric charges. The resulting scalar potential field is used to represent the free space. Collision between the obstacles and the robot are avoided by a repulsive force between them. In this method, path-planning is done in Configuration Space and it also needs a long calculation time.

The number of the dimensions of Configuration Space increases in proportion to that of the manipulator joints. It means that the amount of treated data becomes larger. Then enormous computer memory is needed, or the Configuration Space should be quantized roughly. Considering these reasons the path for manipulator should be planned in the workspace, that is, in the Real-Space. Our method is the path-planning in the Real-Space, which is applicable to multi-axis manipulator.

In order to model 3-D workspace, we use the octree [4] which has been used for fast collision detection. An octree recursively decomposes three-dimensional space into eight equal cubic octans until each octant meets some decomposition criteria.

The levels of Octree are used as the cost function to represent the distance between the manipulator and the obstacles. This criterion is not exact, but, due to this, we can obtain the approximate feasible trajectory extremely quickly.

Section 2 and 3 describe how to model the workspace and the manipulator. The algorithm for path-planning is explained in Section 4. The performance of our algorithm is demonstrated on a variety of examples in Section 5. Section 6 describes conclusion and future work.

2 Model of Environment

In this section, we briefly summarize the use of the octree and how to model the workspace. It usually takes a lot of time to detect collision between the manipulator and the obstacles. We solve this problem by using the octree.

2.1 Octree

We use the octree for collision detection between the manipulator and the obstacles. The octree is a hierarchical data structure which recursively decomposes three-dimensional space into eight equal cubic octans until each

octant meets some decomposition criteria.

First, a given environment is divided into eight equal cubes and check whether each cube collides with the obstacles or not. These cubes are called Octree of *level1*.

If a cube doesn't collide with the obstacles, we need not check collision between the manipulator and the cube of the next level because the cube shows empty space. If a cube collides with the obstacles, the cube is recursively divided into eight equal cubes and collision is checked. This decomposition process is performed until each division satisfies the required resolution.

For example, when we think of an environment including obstacles shown in Figure 1, the process of division are shown in Figure 2. This division should be done once only when the environment is changed.

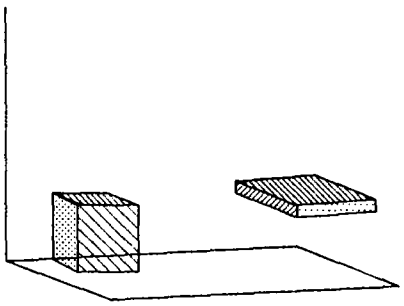

Figure 1: Environment with obstacles.

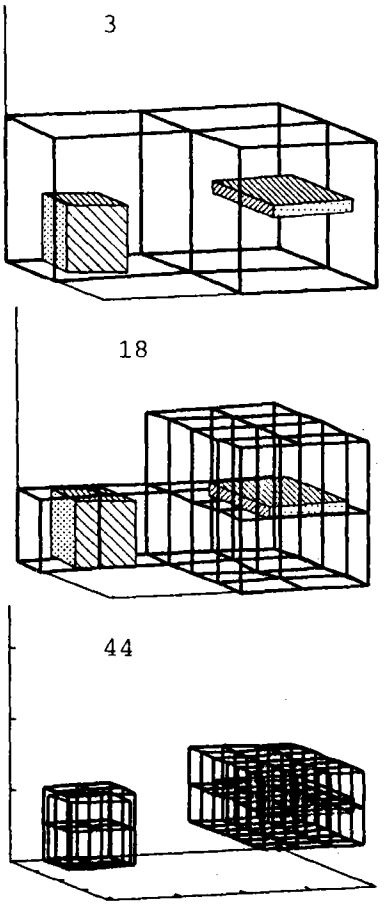

Figure 2: Octree of each level.(The number left above shows the number of cubes which compose that level.)

3 Model of Manipulator

We model the manipulator in a set of cubes because the octree is a set of cubes and with this modeling, we can detect collision against it quickly.

6-axis manipulator are used for explanation and simulation. The manipulator shown in Figure 3 is modeled as Figure 4. Thirty cubes are used to cover it. Each size of cubes are changed to cover the manipulator.

3.1 Collision Detection

This section describes the algorithm to detect the collision. First, we pick up one cube which composes the manipulator and check the collision with the octree of level1 which has eight large cubes. If the octree of level1 and the cube collides, we further check the collides with the octree of level2. These collision checks are done until the cube collides with the octree of last-level or the octree in the obstacles. In this case, we think the manipulator collides with the obstacles.

We repeat these collision check for all cubes that composes model of the manipulator.

These collision check against one manipulator posture needs 2ms in average on an Intel 80486 processor running at 33MHz

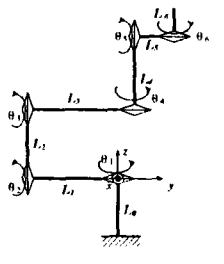

Figure 3: 6-axis robot manipulator

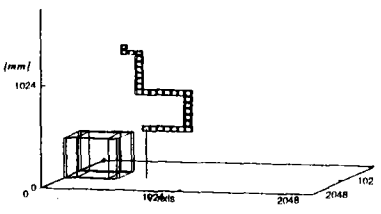

Figure 4: Cubes which cover the robot manipulator

3.2 Manipulator with Baggage

In previous section, we modeled the manipulator in a set of cubes. When the manipulator picks up a baggage with its hand, we need to model the baggage to avoid collision against the obstacles. As complicated modeling makes the planning difficult, it is necessary for the model to be simplified. From my standpoint of view, it is convenient to model the baggage in a set of cubes because they can be treated as a part of the manipulator and need not change the algorithm for path planning whether the manipulator takes the baggage or not.

For example, when a manipulator with baggage as Figure 5 is given, we can model this like Figure 6.

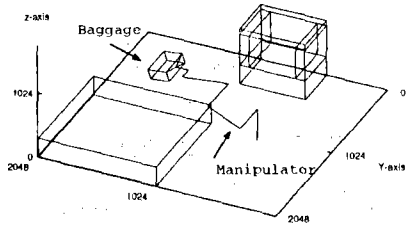

Figure 5: Manipulator with baggage

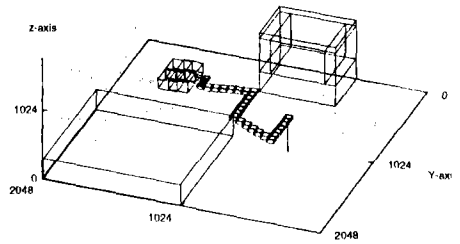

Figure 6: Model of baggage

4 Collision Avoiding Motion

This section describes the algorithm for collision-free path-planning. Using the levels of the octree, we can generate the path quickly.

4.1 Distance among Manipulator and Obstacles

We think that the levels of the octree reflect the distance between the manipulator and the obstacles. In this section, we make a cost function named J_1 which roughly shows how far the manipulator is away from the obstacles.

When we check collision between the manipulator and the obstacles, we can know which level of the octree collides with the each cube of the manipulator and give each cube the value corresponding to its collision level.

Summing up the values of cubes which compose the manipulator, we can make the cost function J_1 which represents distance between the manipulator and the obstacles. The cost function is shown as follows.

$$J_1 = \sum^{manipulator} \begin{pmatrix} cost\ given\ by \\ the\ collision\ level \end{pmatrix} \qquad (1)$$

When the value of J_1 is small, distance between the manipulator and the obstacles is long. To the contrary, when the value of J_1 is big, distance between the manipulator and the obstacles is short.

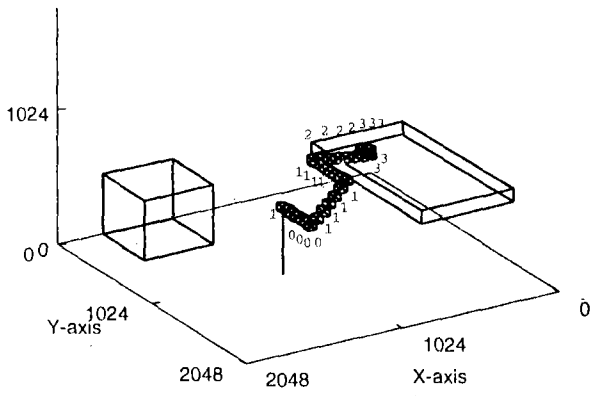

Figure 7: The levels of the octree to detect collision with the manipulator.

4.2 Determination of Manipulator Posture

We make a function which has smaller value when the distance between the manipulator and the obstacles becomes bigger and when the configuration of the manipulator becomes closer to final configuration. If we can get this function, collision-free path can be generated by moving the manipulator in the direction that the value of function becomes smaller. One of such functions is shown as follows.

$$\begin{aligned} J &= J_1 \\ &+ \alpha \sum_6 (CurConf - FinConf)^2 \\ &+ \beta \begin{pmatrix} Distance\ between\ Tip \\ and\ movable\ center \end{pmatrix}^2 \\ \cdots & \quad (2) \end{aligned}$$

Where

$CurConf$	: current configuration.
$FinConf$	: final configuration.
$movable\ center$	: the center of the regions which the manipulator can move freely.

We simplify this equation and write as

$$J = J_1 + \alpha J_2 + \beta J_3 \qquad (3)$$

By setting proper values to parameters α and β, we can get collision-free path by evaluating the function J at each step.

- J_1 drives the manipulator away from the obstacles.
- J_2 is an error between current configuration and final configuration. The manipulator is driven to the final posture as J_2 becomes smaller.
- J_3 is a distance between manipulator top and the movable center shown in Figure 8. This term prevents the manipulators path from growing wider. Even when there are no obstacles in workspace, the region which the manipulator can move freely is strictly restricted. We think the center of such region and pull the path into the direction of the center.

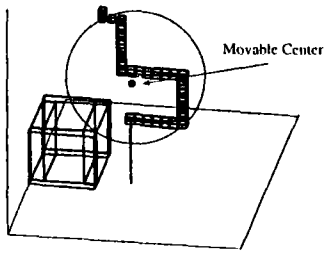

Figure 8: Manipulator can move freely in this cube.

Local minimum of the function J occur sometimes. When local minimum is detected, we increase the value of the parameter α. This means the manipulator is pulled up in the direction of final posture. But further consideration should be given in this point.

5 Path-Planning Examples

Our algorithm has been tested on a variety of examples. First, I show the parameters we set. The max level of the octree is six and the cost for each level of octree is defined as follows.

level1	level2	level3	level4	level5	level6
1	1000	10000	60000	200000	800000

$$\alpha = 1.0$$
$$\beta = 0.5$$

these values also require further consideration. The initial value of parameter α makes no sense because the dimension of function J_1 and J_2 are different. Each joint angle is quantized into seventy two(five degrees).

The algorithm has been implemented on a Sparc Station 10 work station using C language on Solaris Operating System and on a IBM-PC with Intel 80486 processor running at 33 MHz using C language on Linux Operating System.

5.1 Example 1

Collision-free path is planned from the posture shown in Figure 9 to one shown in Figure 10. The result is shown in Figure 11 and execution time is 1.3s with i486(33MHz), 0.8s with SS10.

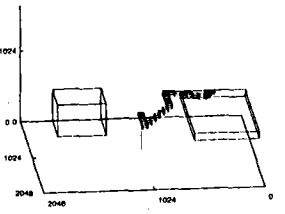

Figure 9: Initial posture

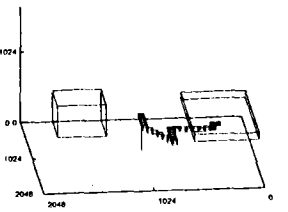

Figure 10: Final posture

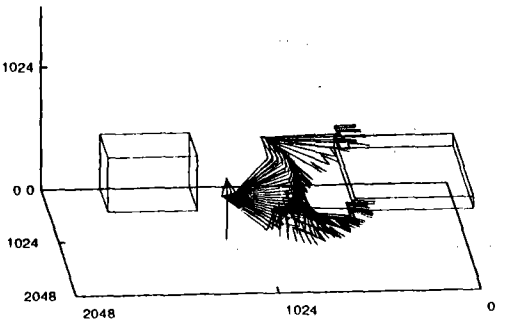

Figure 11: Planned motion(Example 1).

5.2 Example 2

Collision-free path is planned from the posture shown in Figure 12 to one shown in Figure 13. The result is shown in Figure 15 and execution time is 0.8s with i486(33MHz), 0.5s with SS10.

Planning from the posture shown in Figure 12 to the posture shown in Figure 13 and the opposite direction generates different paths. When two postures are given such as Figure 12 and Figure 13, to which direction should we plan the path?

It is not better to plan the path toward the posture shown in Figure 12 from Figure 13 because function J falls into local minimum immediately. So we should plan the path toward the posture shown in Figure 13 from Figure 12, in other words, path planning should be done toward empty space.(shown in Figure 14) We trace backward the planned path when it is necessary.

5.3 Example 3

Collision-free path is planned from the posture shown in Figure 16 to one shown in Figure 17. The result is shown in Figure 18 and execution time is 1.1s with i486(33MHz), 0.7s with SS10.

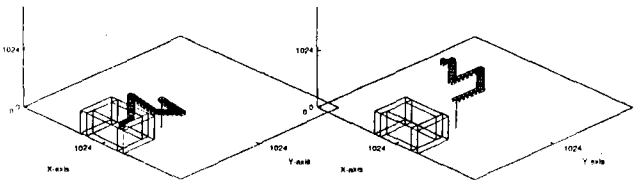

Figure 12: Initial posture

Figure 13: Final posture

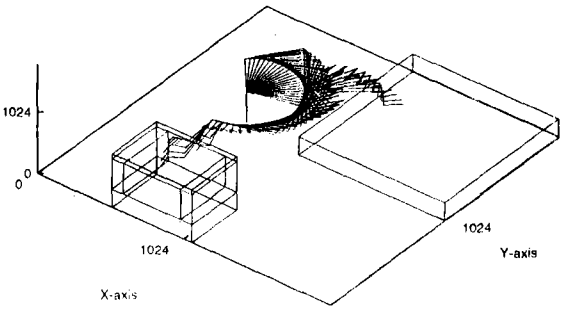

Figure 18: Planned motion(Example 3).

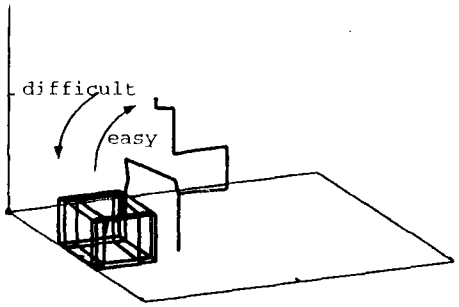

Figure 14: Difference in difficulty due to the motion direction.

5.4 Example 4

Collision-free path is planned from the posture shown in Figure 19 to one shown in Figure 24. This example shows the motion that the manipulator moves the baggage out of the box and put it on the table. The result is shown in Figure 19 $\sim$ Figure 24 and execution time is 3.8s with i486(33MHz), 1.8s with SS10.

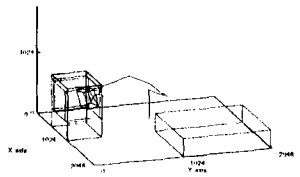

Figure 19: Initial state

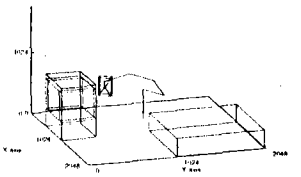

Figure 20: step 1

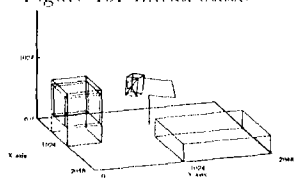

Figure 21: step 2

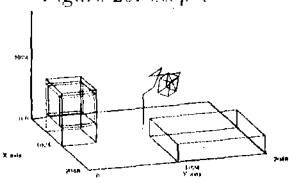

Figure 22: step 3

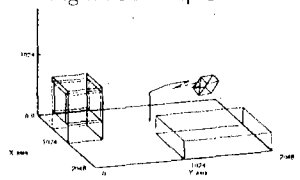

Figure 23: step 4

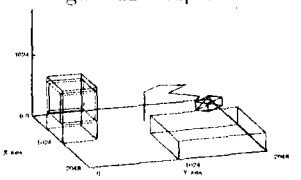

Figure 24: Final state

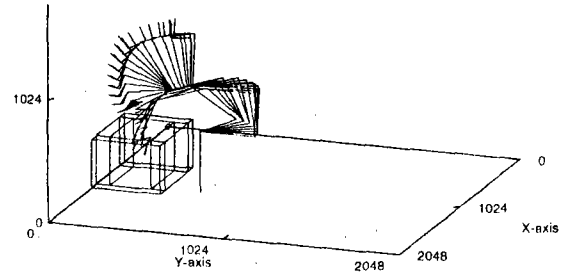

Figure 15: Planned motion(Example 2).

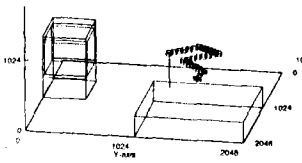

Figure 16: Initial posture

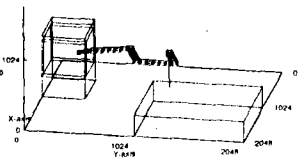

Figure 17: Final posture

5.5 Comparison with other method

The example 4 shown in section 5.4 is the same simulation as the one which was planned using graph search in configuration space and it required about half an hour on VAX11/780 for planning.

Compared with this, we can confirm our method is very quick. (our method requires only 3.8s even with i486.)

6 Conclusions and Future work

In this paper, the method to plan the collision avoiding trajectory quickly for manipulator is proposed.

This method is based on the idea that the levels of the octree reflect the distance between the manipulator and the obstacles.

Our method is applicable to various problems regardless of the number of degrees of freedom of the manipulator, its structure, and the presence of grasped object. This is because the planning is processed not in configuration space but in real space even with multi-axis manipulator.

The experimental results by computer simulation are shown, and the efficiency of our method is confirmed.

We plan to study how to fix the values of parameters and to demonstrate the effectiveness of our algorithm with the real manipulator.

References

[1] Clifford A. Shaffer, Gregory M.Herb, "A Real-Time Robot Arm Collision Avoidance System", IEEE Transactions on Robotics and Automation, Vol.8, No.2, pp.149-160, 1992

[2] Masaaki Shibata, Kouhei Ohnishi, "An Approach to Collision Avoidance Issues For Redundant Manipulator", IECON, pp.1488-1493, 1993

[3] Yonk K. Hwang, Narenda Ahuja, "A Potential Field Approach to Path Planning", IEEE Transactions on Robotics and Automation, Vol.8, No.1, pp.23-31, 1992

[4] Kang Sun and Vladimir Lumelsky, "Path Planning Among Unknown Obstacles, The Case of a Three-Dimensional Cartesian Arm" IEEE Transactions on Robotics and Automation, Vol.8, No.6, pp.776-786, 1992

[5] Hironu Onda, Tsutomu Hasegawa, and Toshihiro Matsui, "Collision Avoidance for a 6-DOF Manipulator Based on Empty Space Analysis of the 3-D Real World", Proc. IROS, pp583-589, 1990

[6] Jin-Oh Kim, Pradeep K.Khosla, "Real-Time Obstacle Avoidance Using Harmonic Potential Functions", IEEE Transactions on Robotics and Automation, Vol.8, No.3, pp.338-349, 1992

[7] Keisuke Sato, "Global Motion Planning using a Laplacian Potential Field", Journal of Robotics Society of Japan, Vol.11, No.5, pp.702-709, 1993

[8] Koichi Kondo, "Collision Avoidance by Free Space Enumeration Using Multiple Search Strategies", Journal of Robotics Society of Japan, Vol.7, No.4, pp.88-98, 1989

[9] H.Noborio, S.Fukuda and S.Arimoto, "A New Interference Check Algorithm Using Octree", Proc. of IEEE International Conference on Robotics and Automation, 1987

[10] Hiroshi Noborio, Motohiko Watanabe and Takeshi Fujii, "A Feasible Algorithm for Planning a Continuous Sequence of Collision-Free Motions of a Manipulator", SICE, Vol.26, No.12, pp95-102, 1990