- A Thesis Submitted for the Degree of Master -

Real-time Collision-free Path Planning for Robot Manipulator Using Octree Model (オクトツリーを用いたロボットマニピュレータのリアルタイム障害物回避軌道計画)

Lu Wu
呉 魯

Supervisor :
Professor Yoichi Hori

Department of Electrical Engineering
The University of Tokyo

2006.2

- A Thesis Submitted for the Degree of Master -

Real-time Collision-free Path Planning for Robot Manipulator Using Octree Model
(オクトツリーを用いたロボットマニピュレータのリアルタイム障害物回避軌道計画)

Lu Wu
呉 魯

Supervisor :
Professor Yoichi Hori

Department of Electrical Engineering
The University of Tokyo
2006.2

Abstract

In this thesis, a real-time path planning approach is proposed. The merit of the approach is very fast and simple. It can be applied to industrial robot working in very changeable environment.

In the approach, Octree model is built according to surrounding. Using Octree model, collision is checked and artificial potential is created. The search in C-space is executed to find a collision-free path. Two factors are considered in the search. One is collision avoidance. Another is task execution. The two factors are embodied in the evaluation function of the search.

The effectiveness of the approach is proved by simulations and experiments.

Moreover, we did comparison between the approach using Octree Model and another quick approach. The quick approach is an approach using A* algorithm. The result shows our approach has advantage in calculation time.

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Professor Yoichi Hori. He not only directs me to carry out my research, but also leads me to search for the philosophy of life. His wisdom and his character of active habit will influence me in all my life. I am very proud of belonging his laboratory from 2004 to 2006, and these two years will be my precious treasure through my life.

I am thankful for the Backing Society for Foreign Student in the University of Tokyo. It gives me financial support for the research.

I wish to thank the University of Tokyo. She provides cheap and nice accommodation for foreign student. She also offers favor for foreign student.

I thank Lianbing Li, Naoki Hata, Nobutaka Bando, Sehoon Oh for helping me on research. I also thank the other people in Hori laboratory for their support.

Finally, I would like to thank my family for their support.

Contents

Chapter 1

Introduction

1.1 Review of the Research

1.1.1 About Path Planning for Robot Manipulator

Path planning of a manipulator refers to find a short and collision-free path from start position to goal position.

Every robotics system requires path planning. Because robots need to avoid collision to execute tasks in a work space.

A manipulator is a robotic arm consisting of links joined together. The configuration parameters are the joint angles, and DOF(degrees of freedom) is equal to the number of independently controlled actuators.

There are a number of advantages to plan the path of robot manipulator automatically. [4]It relieves human workers of the continual burden of detailed path design and collision avoidance, and allows them to concentrate on the robotic tasks at a supervisory level. Robots with an automatic path planning can accomplish tasks with fewer, higher-level operative commands. Robotic tele-operations can then be made much more efficient, as commands can be given to robots at a coarser time interval. With an automatic motion planning and appropriate sensing systems, robots can adapt quickly to unexpected changes in the environment, and be tolerant to modeling errors of the workspace.

However, the path planning for a multi-DOF manipulator is very difficult. It is unlikely to have a uniformly fast path planning capable of finding a collision-free path within a few seconds for all tasks.[5]

1.1.2 Previous Work

(1)The Approach of Graph Search

Most typical approaches of path planning for manipulator are based on graph search in C-space. The C-space denotes spaces that specify positions of any point on the robot uniquely with respect to a coordinate frame fixed in a working space.

Although Lozano-Pe ’rez proposed slice projection approach to describe the entire C-space representation, it is practically difficult to compute slice projection of manipulators with more than 4DOF.

Hasegawa proposed a C-space characterization approach based on arm and hand separately.[3] This approach tries to describe the entire C-space representation prior to path searching. The approach tries to decrease the dimensions of space from 6 dimensions to 3 dimensions.

Kondo pointed out that describing entire C-space representation is waste of time when the environment frequently changes, because it is not always necessary prior to path searching [2]. In his method, collision-freeness is checked only on grid points of the discrete C-space using geometric models while searching for the path. In Kondo 's approach, discrete C-space is regarded as a connectivity graph, whose nodes denote grid points of the C-space and whose arcs/edges denote adjacent relations between the grid points. Then a graph search method called bi-directional heuristic front-to-front algorithm (bi-directional search using A* algorithm) is applied to find paths. Kondo proposed improved performance criterion for A* algorithm by multi-strategic way. Although his method is more effective than another methods needed for the entire C-space representation, its computational cost (time and memory space) increases exponentially as DOF and/or resolution of the discrete C-space increase. This is an inherent problem. To reduce computational costs, both the number of collision-checks and the time required for each collision-check must he reduced drastically.

Chen proposed a hierarchical path search method based on subgoal network.[4] In this method, a global planner constructs a subgoal network. Then, a local planner checks the reachability between subgoals. Global planning and local planning are repeated until either a path is found or no more space exists to search. The subgoal is a sub set of C-space. During this repetition, subgoals are refined, and become a smaller sub set. The primary advantage of this approach is that computational time is proportional to the complexity of the environment and to the difficulty of the task.

Ando succeeds the approach of Chen's.[5] He simplified the approach of Chen's and make calculation more fast. However, the simulation of the approach didn't come into the contact with a 6DOF manipulator. It only involves the calculation of 2DOF and 3DOF manipulator. When the manipulator is 6DOF, the C-space will extend into 6 dimension space. The calculation quantity and the storage of data will be very enormous.

The above approaches try to set up a C-space at first. Then a path planning is executed in this C-space. The problem of these approaches is the high dimensions space when DOF is more than 3.

(2)The Approach of Artificial Potential

The approach of potential is to exert an artificial force graduated by an artificial potential filed. The potential is like an electrical potential or gravity potential, but it is an artificial potential. This approach is first assumed by Khatib[8]. It is a very interesting approach of path planning.

In this artificial potential field, there is a high value of potential on the side of obstacles. And it slopes down toward the goal configuration, so that a manipulator can reach the goal configuration from any other configuration by following the negative gradient of the potential. The high value of the objects potential prevents a manipulator from running into obstacles.

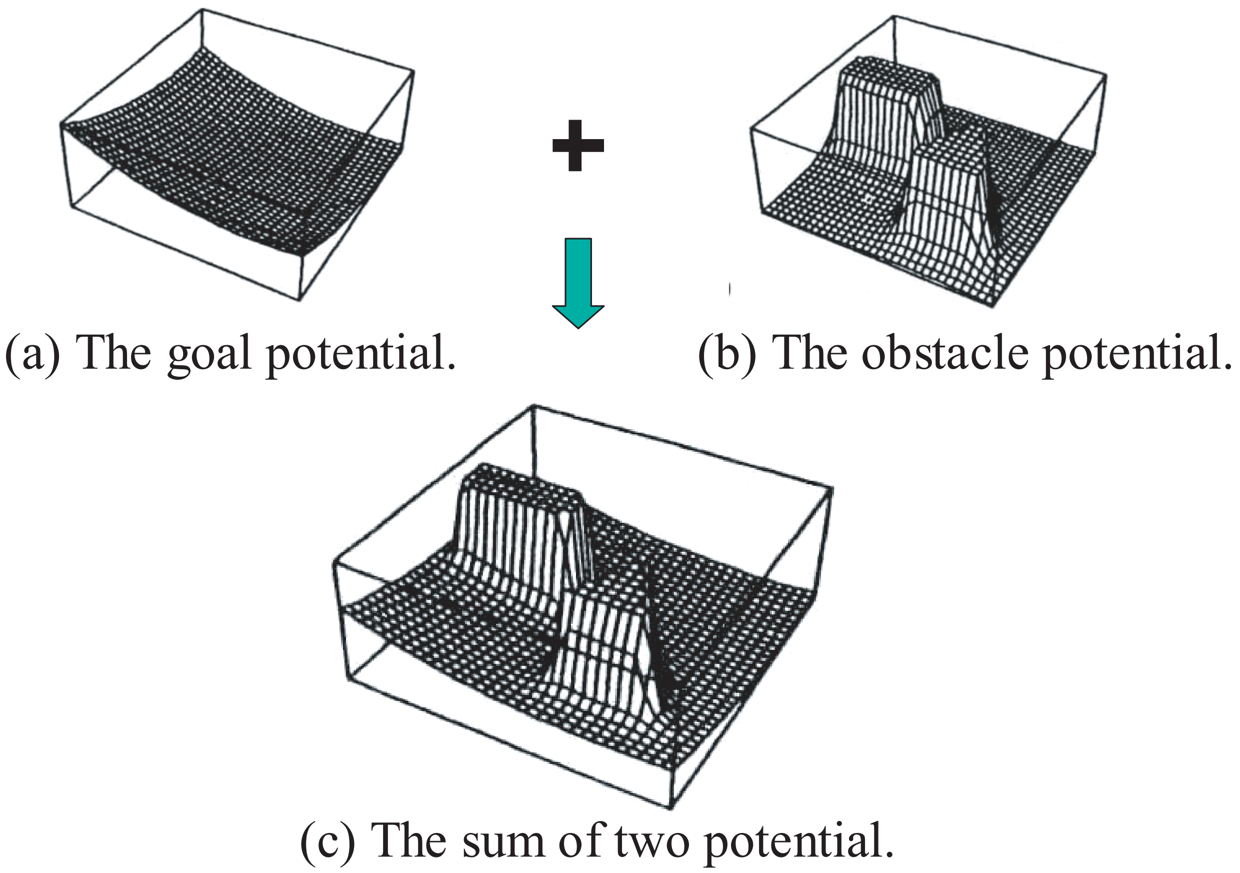

Fig. 1.1: The graph of artificial potential.

The sum of two artificial is shown as the following equation:

$$U_{sum}(q) = U_{goal}(q) + U_{obstacle}(q) \tag{1.1}$$

And the graduating force is the gradient of the potentials:

$$F_{sum}(q) = -grad[U_{sum}(q)] \tag{1.2}$$

Fig. 1.1 illustrates this approach. First, an goal potential is constructed that has a high value on the side of point of start and low value on the side of goal. (Fig. 1.1 a) The negative of the inverse of distance to the goal is a commonly used as goal potential. Then the potential of obstacle is constructed.(Fig. 1.1 b) The inverse of distance to obstacles can be used for this purpose. The force generated by the obstacle potential makes a manipulator avoid collision with the obstacles. The force generated by the goal potential makes the manipulator move toward the goal. The sum of the two potentials is computed (Fig. 1.1c), and the moving path from the start to the goal avoiding collision is made by the artificial potential.

(3)The Approach of Roadmap

The approach of roadmap is a new motion planning method for robots in static workspaces. This method proceeds in two phases: a learning phase and a query phase. In the learning phase, a probabilistic roadmap is constructed and stored as a graph whose nodes correspond to collision-free configurations and whose edges correspond to feasible paths between these configurations. These paths are computed using a simple and fast local planner. In the query phase, any given start and goal configurations of the robot are connected to two nodes of the roadmap; the roadmap is then searched for a path joining these two nodes. The method is general and easy to implement. It can be applied to virtually any type of holonomic robot. A holonomic robot is capable of rotating (turning) while moving forward or backward and left or right. In other words, it is a 3 degree-of-freedom base, which is the maximum possible degrees-of-freedom for a mobile robot. One impressive feature of a holonomic robot is that it can rotate while moving in a straight line.

1.2 Objective of the Research

As shown above, a lot of research has been devoted to the issue of path planning for manipulator. However, few algorithm of path planning is applied in practical industry because this problem becomes complex and time-consuming as a many-DOF robot manipulator executes task in cluttered environments. In factory, collision-free trajectory for industrial robot is usually done by human. There are few practical path planners that is simple enough and quick enough.

On the other hand, many of future robot tasks, such as assembly and disassembly, teleoperation, and medical surgery, will be executed in dynamic environments. Therefore, a real-time path planner for many-DOF manipulator is necessary. A robot with manipulator can not react to dynamic changes in environment if it has not the capability of real-time path planning.

For the above reasons, the paper proposes a real-time approach to generate path for robot manipulator.

1.3 The Outline of the Thesis

This thesis is composed by two approaches. One is the approach using Octree model. Another is the approach using A* algorithm. They are stated separately in Chapter 2, and Chapter 3.

The content of every chapter is shown below.

In Chapter 1 the research of path planning will be reviewed. Objective of the research will be stated and the outline of the paper will be introduced.

In Chapter 2, the approach based on Octree model is illustrated carefully. Moreover the simulation and experiment will be stated.

In Chapter 3, a typical graph search approach is stated and its computation result will be compared with the result of the approach using Octree.

The paper is concluded in Chapter 4.

Chapter 2

The Approach Based on Octree Model

2.1 The Outline of the Approach

The approach in this chapter, 'path planning based on Octree model', has composite character. Generally, the approach has three main features. The first one is to use Octree model. The second one is artificial potential. The third one is to search for finding a path. They are stated as bellow.

2.1.1 Octree Model

Octree is used to transform surrounding information into computer. Octree divides space into subspaces smaller and smaller. A computer will calculate and store the data of every subspace. Based on Octree model, surrounding space is illustrated in computer.

In addition, the distance between manipulator and obstacle is estimated by generating artificial potential. When the value of artificial potential is small, the distance between manipulator and obstacle is long. When the value of artificial potential is big, the distance between manipulator and obstacle is short.

Moreover, collision detection is done by Octree model. The points on manipulator are checked whether there is obstacle in the highest level of Octree model. If there is obstacle, collision is identified.

2.1.2 Artificial Potential

The potential is like an electrical potential or gravity potential, but it is an artificial potential. After generating the potential, robot will be repelled by obstacle, and be attracted by goal.

Based on Octree model, artificial potential is generated. The points on the surface of manipulator are taken. Using kinematics, the positions of the points are calculated. Then check whether there is obstacle in the space of this level. The checks are done from the lowest level until the highest level. If there is obstacle in the space of this level, potential is generated.

2.1.3 Search

Path planning is executed by search step by step in C-space. Which step will be taken is decided by the action of artificial potential.

In every step, there are (2×n-1) kinds of possibility. n is DOF of manipulator. The evaluation function will be calculated separately for every kind of possibility. The best step will be selected by the value of the evaluation function.

2.1.4 Algorithm Flow

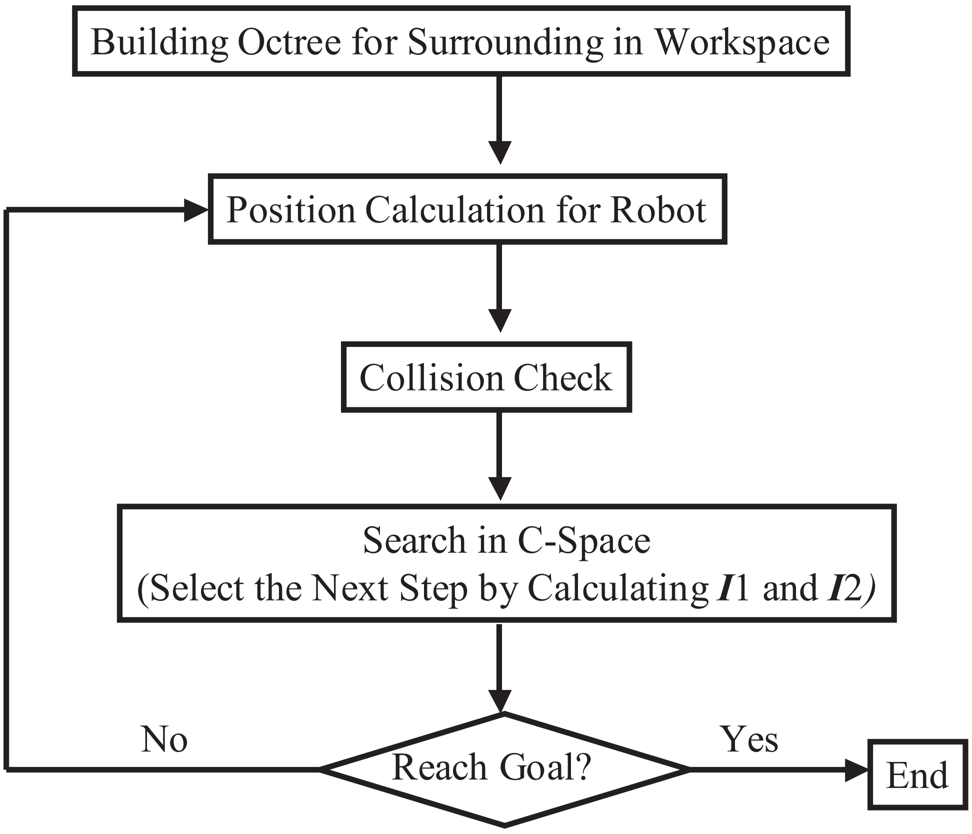
Building Octree for Surrounding in Workspace
Position Calculation for Robot
Collision Check
Search in C-Space
(Select the Next Step by Calculating *I*1 and *I*2)
No
Reach Goal?
Yes
End

Fig. 2.1: The algorithm flow diagram.

The algorithm flow of the approach can be stated as:

1. Building Octree model.
2. Calculate the coordinate of the points on manipulator.
3. Decide which nodes the points are in. Check collision.
4. Calculate the value of estimation function. Decide which step should be taken.
5. Return to Step 2 until attain the goal.

The algorithm flow diagram is shown in Fig. 2.1.

2.2 Octree Model

2.2.1 The Introduction of Octree Model

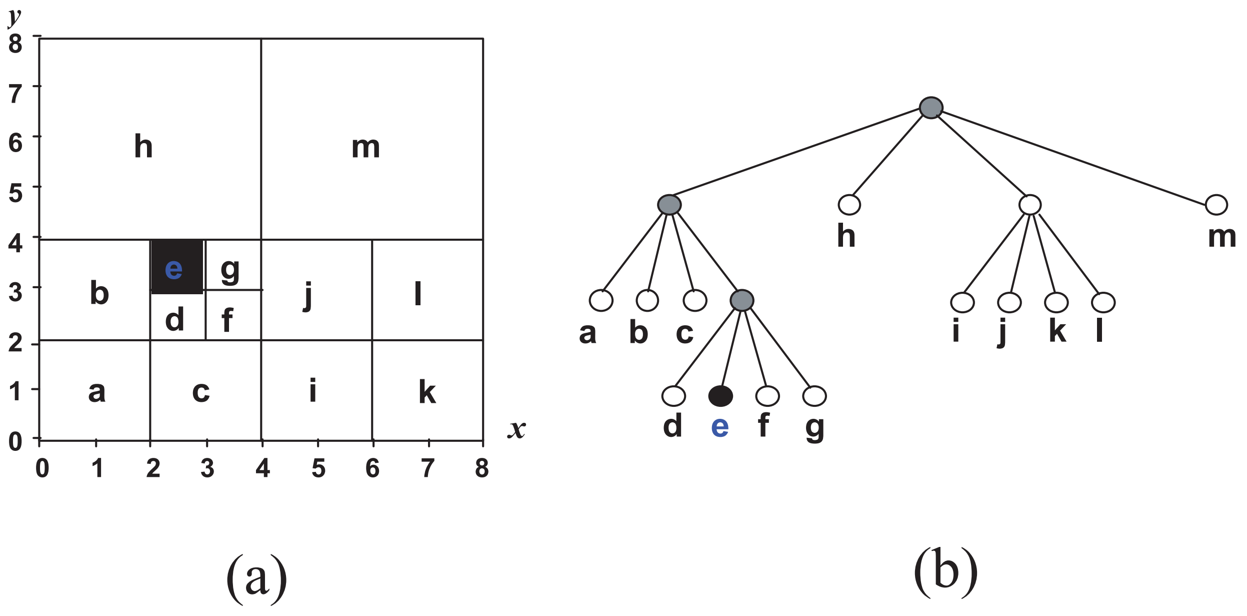

Fig. 2.2: The Model of Octree in 2D space. (a) is the space divided by Octree. (b) is the structure of the data stored.

Octree is a decomposition model of space. Octree is a compact and memory efficient hierarchical structure for representing three dimensional spatial occupancy data. It usually needs only a small fraction of memory to represent a set of space. It's an efficient operation algorithm and is attractive to many applications in robotics, computer vision and other fields.

Fig. 2.2 shows Octree model in plane space (Octree in 2D space is also called Quadtree). Octree divides space into subspaces smaller and smaller. Every time space is divided into 4 blocks. It will be checked whether there is obstacle in every block. For example, the block ' e ' is a subspace divided as shown in the figure. If the subspace is full of the obstacle, the color of the node ' e 'is black. In (b) of Fig. 2.2, the parent node of this node is gray, expressing the block is mixing.

Fig. 2.3 shows Octree model in 3D space. The space is divided into 8 sections every time by Octree. In the model which is shown in the figure, a cell is represented by black color , by white color or by in gray color. A set of black cells corresponds to a set of space full of object. And the white ones correspond empty. A grey one corresponds mix space with both object and empty space.

Why use Octree? Because surrounding needs to be let a computer know. Octree can express the surrounding and translate the information of the surrounding into the computer.

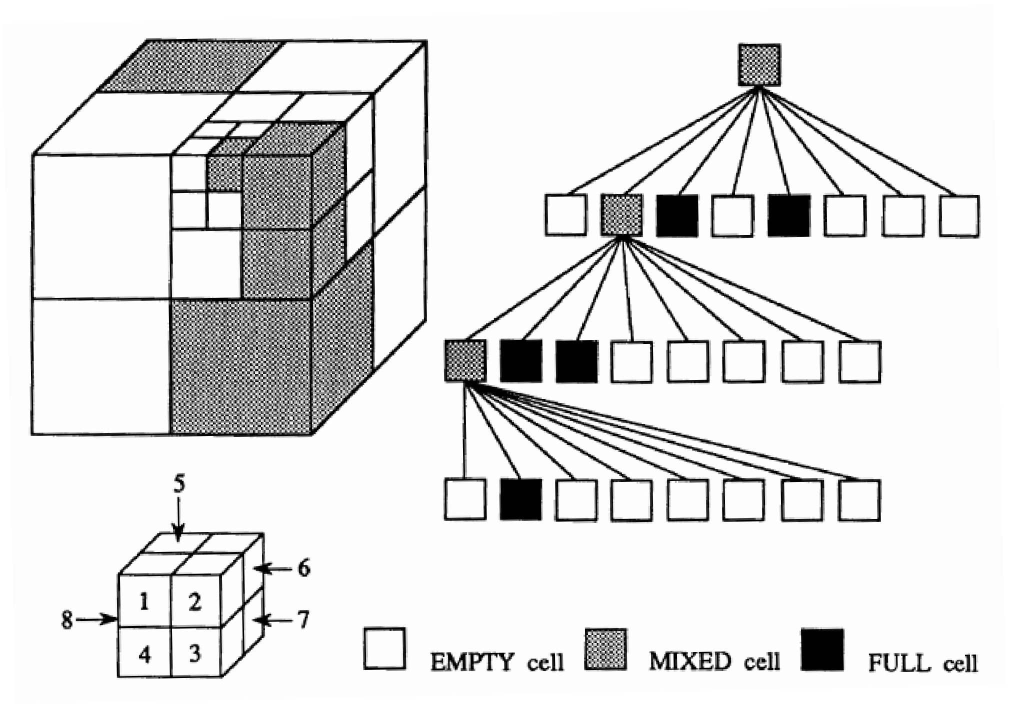

Fig. 2.3: The Model of Octree in 3-D space.

This is a reason of using Octree.

Another reason using Octree is to decrease the calculation quantity and to save memory. In the case of a 6-DOF manipulator, it needs much more resource of a computer for a C-space modeling.

2.2.2 The Action of Octree Model

There are two actions of Octree model. The first one is Octree is used to generate artificial potential. A point in space is checked which level it is in Octree model. In different level, different artificial potential will be generated. In higher level of Octree model, it means that the point is nearer to the obstacle. The potential will be bigger. In lower level of Octree model, it means that the point is further to the obstacle. The potential will be less.

The second one is Octree can be acted as collision checker. A point in space will be checked until to the highest level of Octree model. If the point is in the highest level of Octree model, it can be judged that a collision has happened.

2.2.3 Building Octree

Building Octree is to translate coordinate to Octree code. The calculation method to build Octree is expressed as below:

1. x, y, and z of a point divide the length, the width, and the height of space.
2. The values compare with 0.5. If it is greater than 0.5, the result is 1. If it is less than

0.5, the result is 0. Because there are x, y, z, three coordinates, the result will be a binary number, such as 111. Then the decimal code of the first level Octree will be 4+2+1=7.

3. Then the value divided in step 1 will be compared with 0.25(or 0.75). If it is greater than 0.25(or 0.75), the result is 1. If it is less than 0.25(or 0.75), the result is 0. Then the binary calculation will be done to get the code of the second level Octree.

4. The calculation will be done until the last level.

For example, in three level Octree, there is a point of obstacle, whose coordinate is (60,70,60)cm. Its coding will be 702. The number 7 means the point is in the 7th node of the first level. The number 0 is the position in the next level Octree. It means the position is the 0th node in the second level Octree. The number 2 means the position is in the 2nd node in the third level Octree.

The calculation is described as below. Assume the length, the width, and the height of the whole space are 100cm. The ratio are $\frac{60}{100} = 0.6$, $\frac{70}{100} = 0.7$, $\frac{60}{100} = 0.6$. Because the ratios are all bigger than 0.5, the coding of level 1 are 111. The result is 4+2+1=7 after binary calculation. Then, because 0.6 and 0.7 are both less than 0.75, the result of binary code is 000. Then the result of decimal code is 0. In the next level, 0.6 is less than 0.625, 0.7 is bigger than 0.625, and less than 0.75. Then the binary code is 010. So the decimal code in the third level Octree is 2. Therefore, the point (60cm,70cm,60cm) can be stored as a decimal code 702.

2.3 The Calculation on Robot

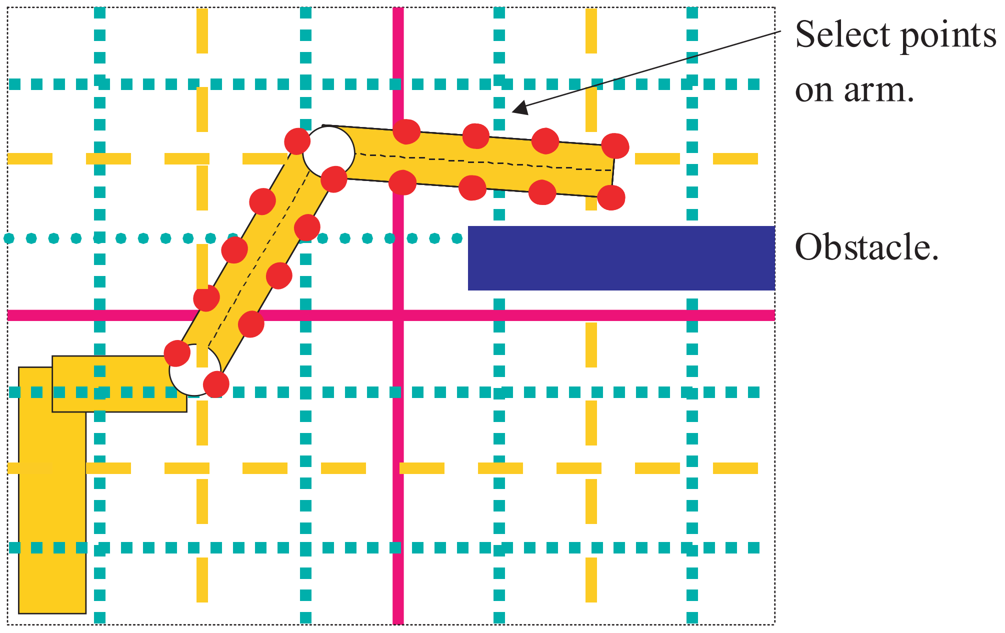

Fig. 2.4: A two-link manipulator in 2-D space which is divided by Octree. In the figure, the solid lines divide the space into level 1. The chain lines divide the space into level 2. The dot lines divide the space into level 3.

2.3.1 The Model of Robot

The information of surrounding is transformed into computer by the method stated in 2.2.3. The information of robot is also expressed by Octree model.

The model of robot is built by selecting the points on the surface of robot. Fig. 2.3 shows the initial state and the end state of Fanuc Robot(LR Mate 200iB) in a side view. Because the arms of the FANUC robot are thick, they can not be seen as lines. Therefore, the points on the surface of them are taken to express the arms of the robot. And calculate the coordinate of these points, decide which nodes of the Octree they are in. Then compare with the nodes which are filled with obstacle, to estimate the distance between manipulator and obstacle.

2.3.2 Kinematics Calculation

Kinematics calculation means that joint angles are given, to find the position of robot arm. In order to calculate the coordinate of the points on the center lines of arms, kinematics calculation is necessary. The parameters of FANUC robot, LR Mate 200iB is shown in Table 2.1.

Parameter / Link	θ_i(Degrees)	d_i(m)	a_i(m)	β_i(Degrees)
1	$\theta 1$	0.35	0.15	90
2	$\theta 2$	-0.154	0.25	0
3	$\theta 3$+90	0.154	0.075	90
4	$\theta 4$	0.29	0	90
5	$\theta 5$	0	0	-90
6	$\theta 6$	0.08	0	0

Table 2.1: The parameters of FANUC robot, LR Mate 200iB.

In this table:

θ_i is the joint angle from x_{i-1} axis to x_i axis about Z_{i-1} axis(using the right hand rule).

d_i is the distance from the origin of the (i-1)th coordinate frame to the intersection of Z_{i-1} axis with x_i axis along Z_{i-1} axis.

a_i is the offset distance from the intersection of Z_{i-1} with x_i axis to the origin of the ith frame along x_i axis(or the shortest distance between the Z_{i-1} and Z_i axis).

β_i is the offset angle from Z_{i-1} axis to Z_i axis about x_i axis(using right-hand rule).

Using equation 2.1, the coordinate in one coordinate system can be transformed into another coordinate system.

$$^{i-1}\mathbf{A}_i = \mathbf{Rot}(z,\theta)\mathbf{Tran}(z,d)\mathbf{Tran}(x,a)\mathbf{Rot}(x,\alpha) \tag{2.1}$$

In the equation:

${}^{i-1}\mathbf{A}_i$ is the transformation matrix from the $(i-1)$th link to the ith link. **Rot**('axis', 'angle') is the transformation matrix after rotating 'angle' about the 'axis'. **Tran**('axis', 'distance') is the transformation matrix after moving 'distance' along the 'axis'.

After calculation, ${}^{i-1}\mathbf{A}_i$ can be expressed by equation 2.2.

$$\mathbf{{}^{i-1}A_i} = \begin{pmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & a_i cos\theta_i \\ cos\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i cos\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.2}$$

The coordinate of end-effecter can be calculated by equation 3.4.

$$\begin{aligned} \mathbf{T} &= {}^0\mathbf{A}_1{}^1\mathbf{A}_2{}^2\mathbf{A}_3{}^3\mathbf{A}_4{}^4\mathbf{A}_5{}^5\mathbf{A}_6 \\ &= \begin{pmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{2.3}$$

The coordinate of end-effector is (p_x,p_y,p_z).

Fig. 2.5 shows a state of the robot and every coordinate system of the 6 joints.

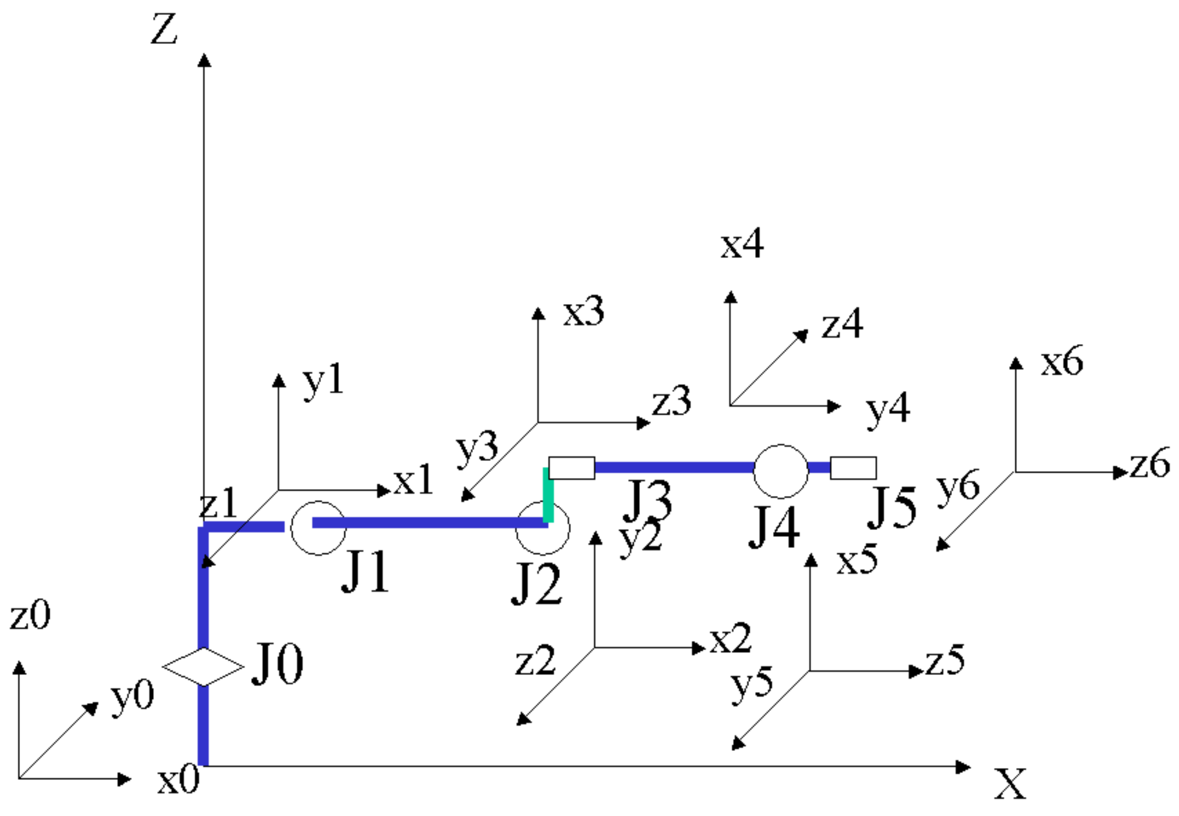

Fig. 2.5: A state of the robot and the coordinate systems of joints.

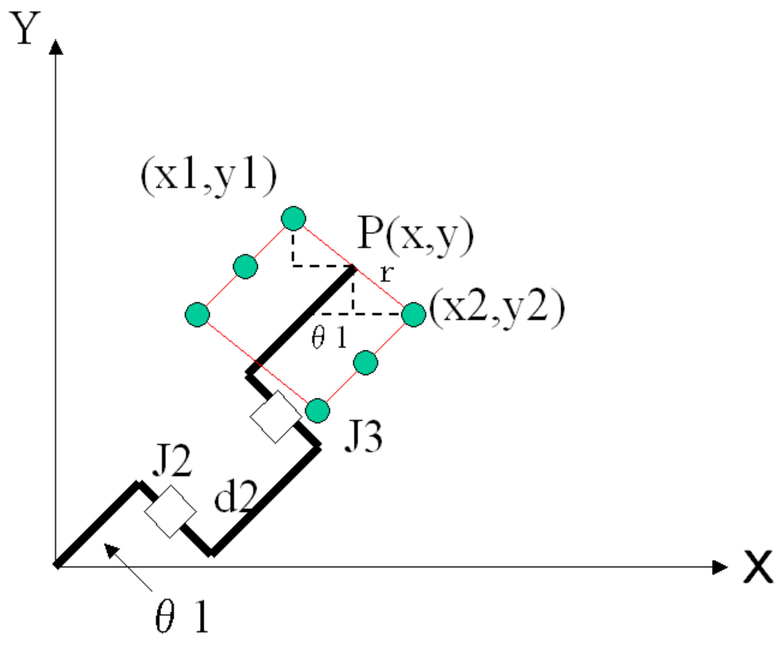

Fig. 2.6: Top view for coordinate calculation.

2.3.3 The Calculation of the Points on Robot

By kinematics calculation, the coordinate of the points on arm's center lines can be calculated. the coordinate of the points on surface need be calculated.

The arms can be recognized as a cylinders. The points on the edges can be calculated by geometry method. Fig. 2.6 and Fig. 2.7 are the top view, and the side view of arms. In the figures, the bold lines are the center lines of robot arm. The points on the surface of robot arm are selected. 3 points on one surface are expressed in the figures. In the approach, there are more points to be selected.

The coordinate of the point on side edge can be calculated by

$$x2 = x + r * sin\theta 1 \tag{2.4}$$

$$y2 = y - r * cos\theta 1 \tag{2.5}$$

In the equations, (x,y) is the coordinate of joint. (x2, y2) is the coordinate of the point on the edge. z2 is equal to the coordinate of the joint. r is the radius of the cylinder we recognize an arm as. $\theta 1$ is the angle of joint1.

The coordinate of the point on top edge can be calculated by

$$z1 = z - r * cos\alpha \tag{2.6}$$

In this equation, α is the angle between robot arm and x axis. (x1,y1) can be calculated in the top view plane.

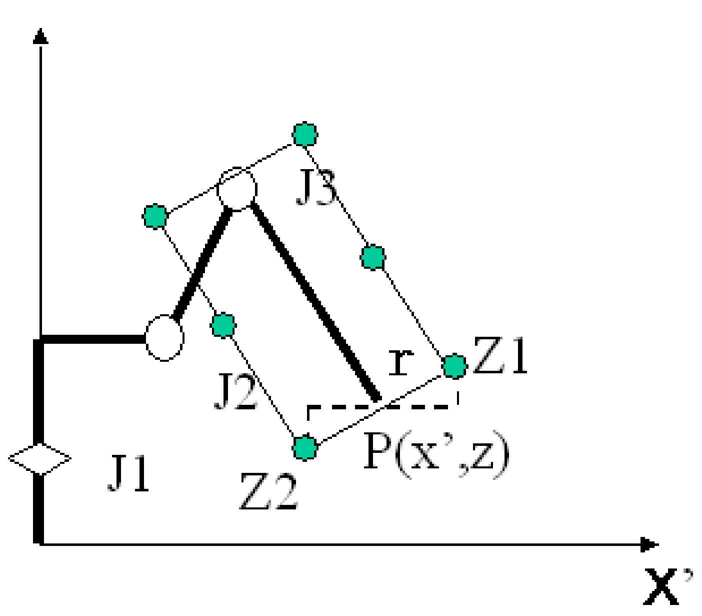

Fig. 2.7: Side view for coordinate calculation.

2.4 Search Algorithm

2.4.1 Evaluation Function

There are two parts in the evaluation function, $I1$ and $I2$.

$I1$ is the sum of the potential between the every point on arm and obstacle.

$$I1 = \sum_{Every\ point\ on\ arms.} (Potential) \tag{2.7}$$

When the collision between manipulator and obstacles is checked, the potential corresponding to its collision level will generate, which repels arms from obstacle. The values corresponding to the collision level are shown below.

level 1: 2^0
level 2: 2^1
level 3: 2^2
level 4: 2^3
level 5: 2^4
level 6: 2^5

The calculation method can be concluded by equation 2.8:

$$P = 2^{n-1} \tag{2.8}$$

In this equation, P means the potential generated by Octree model. n means the level of Octree model.

Summing up the values of the points which compose a manipulator, $I1$ can be calculated. The value of $I1$ can represents the inverse of the distance between the manipulator and the obstacle. When the value of $I1$ is small, the distance between manipulator and obstacle is long. When the value of $I1$ is big, the distance between manipulator and obstacle is short.

Why $P = 2^{n-1}$?

Why is the evaluation of $I1$ calculated by this equation? Because $I1$ can be recognized as the potential generated by obstacle. The potential can be related with the *Distance* between manipulator and obstacle, by calculating equation 2.9:

$$I1 = \frac{k}{Distance} \tag{2.9}$$

In the equation, *Distance* means the distance between manipulator and obstacle. Its inverse function $I1$ can be acted as the potential of obstacle, which expresses the rejection of obstacle.

The *Distance* can be estimated by Octree model. When a point is in level 1 of Octree, the *Distance* can be estimated as 1. If it is in level 2, the *Distance* can be estimated as 2^{-1}. Fig. 2.8 can show the estimation.

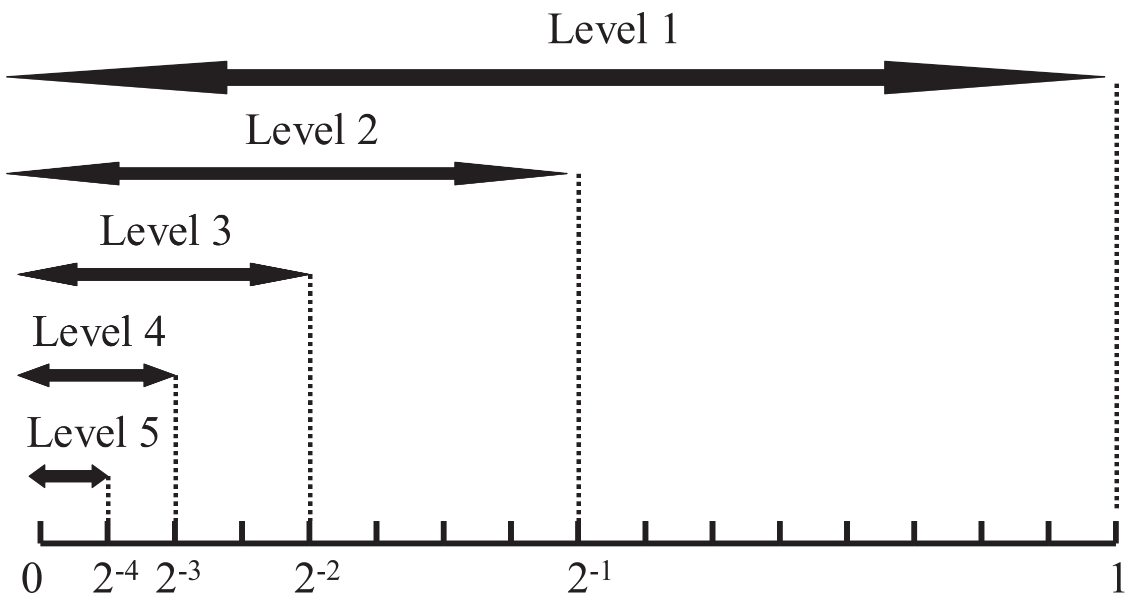

Fig. 2.8: Estimate *Distance* by Octree model.

Then the *Distance* can be estimated by:
level 1: 2^0
level 2: 2^{-1}
level 3: 2^{-2}
level 4: 2^{-3}
level 5: 2^{-4}

level 6: 2^{-5}

The estimation can be calculated by equation 2.10.

$$Distance = 2^{-(n-1)} \tag{2.10}$$

Its inverse function is $I1$. Therefore, $p = \frac{k}{Distance} = 2^{n-1}$(when k=1). This is the reason of equation 2.8.

$I2$ is the and error between the current configuration and the goal configuration. It can be an equation like:

$$I2 = \sum_{DOF} \alpha_i(\theta_i - Goal_i)^2 \tag{2.11}$$

In the equation, α_i is the weight of $I2$ for every joint. It will be discussed in section 2.5.1 and 2.5.2. θ_i is the current configuration of the ith joint. $Goal_i$ is the goal configuration of the joint.

$I2$ is the 'drive strength' that make manipulator go to goal configuration. The value of $I2$ is less, the configuration to goal configuration is close. In other words, manipulator will close to goal configuration when less $I2$ is selected.

Then the evaluation function can be written as:

$$I = I1 + I2 \tag{2.12}$$

In a joint space, the path of the search is decided by the value of the evaluation function at each step. $I1$ drives the manipulator away from the obstacle. $I2$ drives the manipulator to the final posture.

2.4.2 C-space

The work space refers to the physical space robots and obstacles exist in. C-space (the abbreviation of 'configuration space') is the space of all possible configurations of a robot. C-space is a space that specifies any configuration of a robot uniquely using a point in this space. It represents all possible state of robot and plays a fundamental role in path planning.

The C-space parameters of a manipulator are the angles of each joints. Configruation obstacle is obstacle areas in a C-space. The empty area of a C-space is called free C-space.

The Fig. 2.9 illustrates the difference between workspace and C-space of a 2-link manipulator.

Why C-space is defined and used? The first reason is the redundancy of a multi-DOF manipulator can not specify the manipulator's posture completely in a work space. The second reason is one point in work space can tell us only one point in a manipulator. But a point in C-space can tell us all the information of a manipulator.

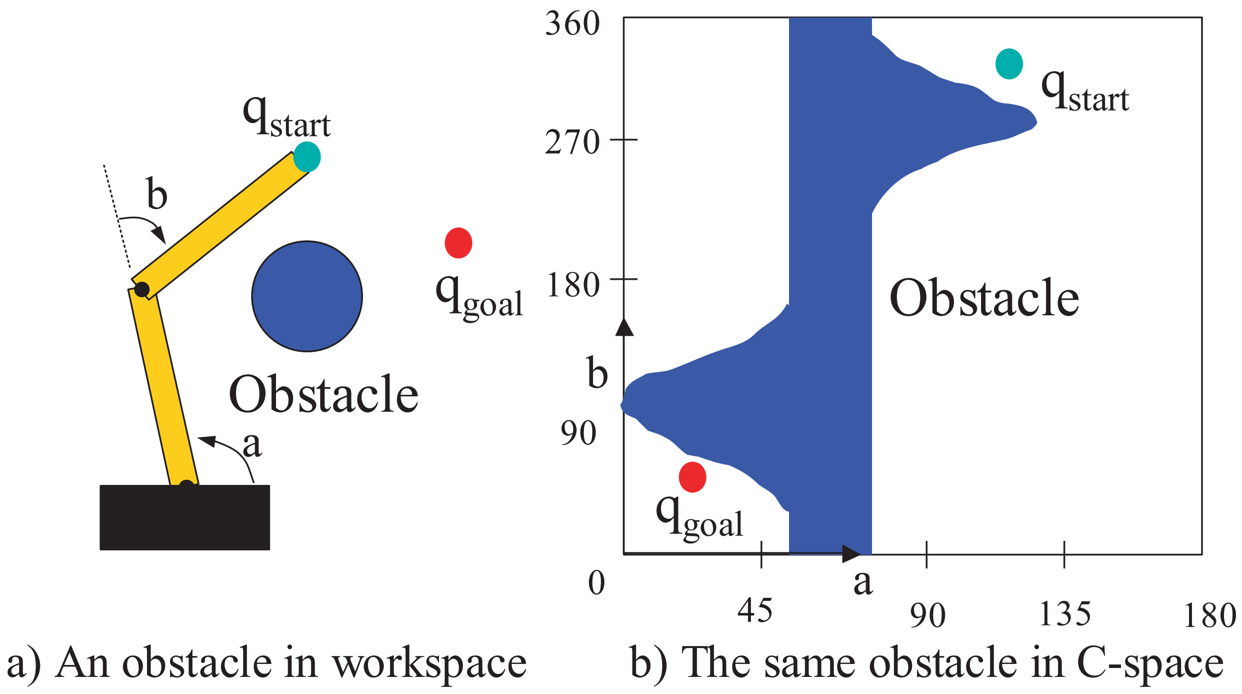

Fig. 2.9: Work space and C-space of 2-link manipulators.

However, if the robot's manipulators are more than 3 degrees of freedom, the C-space will be a multi-dimensions space which is very complex. Some research on path planning is not based on C-space, but is based on work space.

2.4.3 The Process of Search

Fig. 2.10 shows the search method for 2-DOF manipulator. In every step, there are 3 kinds of possibility. In the state of the point O, they are OA, OB, OC. (The step of OD is the return of last step.) The evaluation function will be calculated separately for OA, OB, and OC. The best step will be selected by the value of the evaluation function. There will be 11 kinds of possibility in the case of a 6-DOF manipulator.

Then it can be concluded that the search possibility of every step in the approach is $(2 \times n - 1)$. But the possibility is $3^n - 1$ in the approach using A*. (n is the number of DOF.)

Why $(2 \times n - 1)$**? Why** $(3^n - 1)$**?**

Why the search possibility of every step is separately $(2 \times n - 1)$ and $3^n - 1$ for the two approaches? It can be seen from the view point of every joint's action.

In this approach, only one joint rotates in one step. The joint rotates towards positive direction, or towards negative direction. Then there are 2 kinds of possibility for one joint. If this joint doesn't rotate and another joint rotate, then more 2 kinds of possibility are

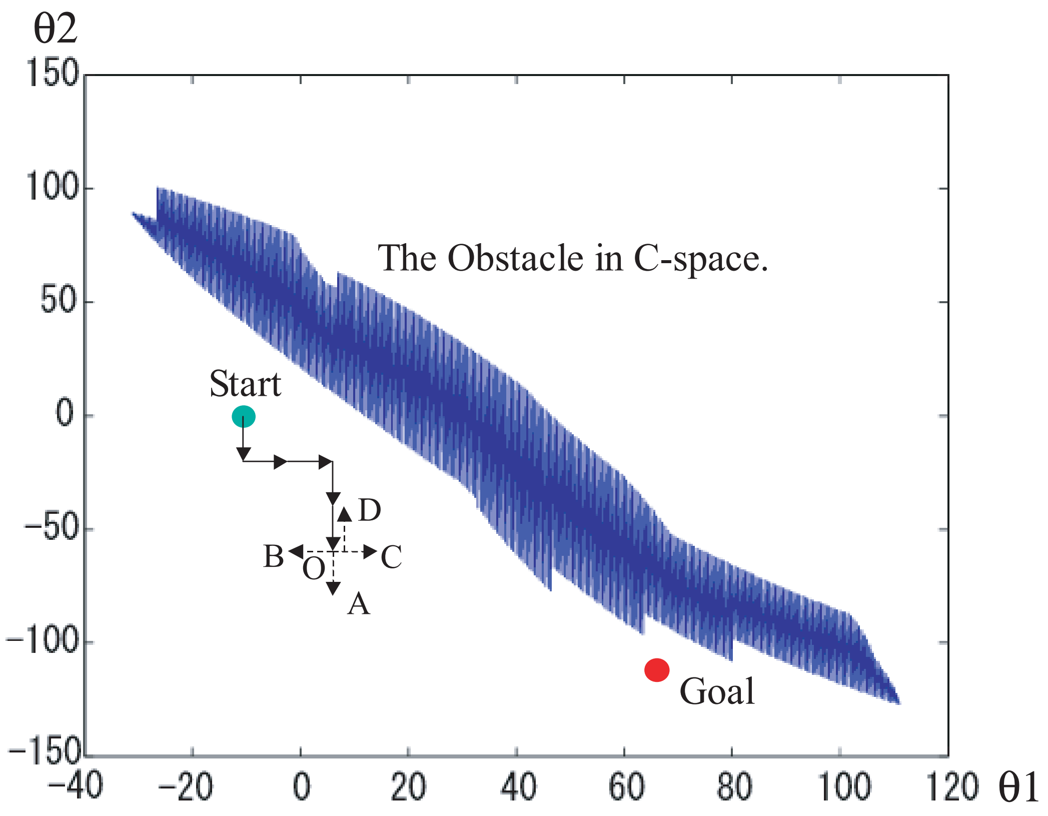

Fig. 2.10: Search process.

added. Because there are also 2 kinds of possibility for the joint, one is the rotation towards positive direction, another is the rotation towards negative direction. Then plus all joints, the value is $2 \times n$. Moreover, the step that lead to go back can not be selected. Therefore, the possibility is $(2 \times n - 1)$.

In the case of the approach using A* algorithm, every joint can be rotatory in one step. Moreover, every joint can be static. Then there are 3 kinds of possibility for one joint. One is the rotation towards positive direction, another is the rotation towards negative direction, the third is static state. Therefore, the possibility is 3^n for all n joints. The algorithm is multiplication because every joint can rotate in one step this time. But one state is impossible. It is that all joint is static. Therefore the possibility is $3^n - 1$.

2.5 Discussion on the Parameters

In the case of two joints, the evaluation function is

$$I = I1 + I2 = I1 + \alpha_1(\theta_1 - Goal_1)^2 + \alpha_2(\theta_1 - Goal_2)^2 \tag{2.13}$$

α_1 and α_2 are two important parameter which will decide the success of search.

2.5.1 Relation between $\alpha1$ and $\alpha2$

The ratio of $\alpha1$ and $\alpha2$ decides flexibility of joint. For example, When $\alpha1 >> \alpha2$, joint 2 can be very flexible in path planning. In example 1, the initial angle of Joint 2 is 0 degree, the object angle is also 0 degree. But joint 2 changes flexibly in the path planning. It changes from 0 degree to about 90 degrees in the process of search, and changes from about 90 degrees to 0 degree again in the end. If $\alpha1 \cong \alpha2$, for example, let $\alpha1$=0.05, $\alpha1$=0.045, the path planning will fail. As shown in (a) of Fig. 2.11, the arm collided the obstacle in the path.

Joint 2 will change a little, because $\alpha2$ is big. The path planning will be failure.

Therefore, the ratio of $\alpha1$ and $\alpha2$ will decide the flexibility of the joint. If a joint is hoped to change great to avoid the obstacle, α of the joint should be given a smaller value in comparison with α of other joints.

2.5.2 The Decision of α_i

In the case of 2-DOF, only $\alpha1$ and $\alpha2$ are included. In the case of many-DOF, $\alpha1$, $\alpha2$, ..., α_i are included. According to the statement above, the ratio among α_i is important. In other words, the decision of α_i is important.

The general rule for the decision of α_i is based on what is stated above. In the algorithm, α of the joint that need be flexible to avid obstacle will be set smaller than other joints'. (For example it is set as 1/10 of other joints'.) Then the joint can be more flexible than the others to rotate in search for avoiding obstacle. Usually the joint which need be flexible is joint 2 or joint 3.

2.5.3 Relation between $I1$ and $I2$

It is better for considering $I1$ and $I2$ as two kinks of strength rather than considering as cost function. Like two kinds of strength, $I1$ and $I2$ pull the arms from start to goal, avoiding collision. $I1$ rejects the arms from the obstacle. $I2$ attracts the arms to the goal.

When $I1>>I2$ (α_1, α_2 are very little), the strength of attraction will be very little, making the arms abandon the search in the halfway. The phenomenon is that the path of search recycles. Therefore the condition of $I1 \cong I2$ is necessary for the search. In the algorithm at present, when $I1/I2>5$, $\alpha1$ and $\alpha2$ will be multiplied 1.3.

When α_1, α_2 are too little, the arms will strike the obstacle at last. The result of the path(α_1, α_2 are 0.0008, 0.00007) is shown as (b) in Fig. 2.11.

When $I2>>I1$, the strength of attraction will be so big that let the arms neglect the rejection of obstacle, striking the obstacle.

An example is cited here. If α_1, α_2 in the first example are 10, 0.9, the result of the path will be shown as (c) in Fig. 2.11. As shown in the figure, the arms will neglect the rejection of the obstacle and collide with the obstacle.

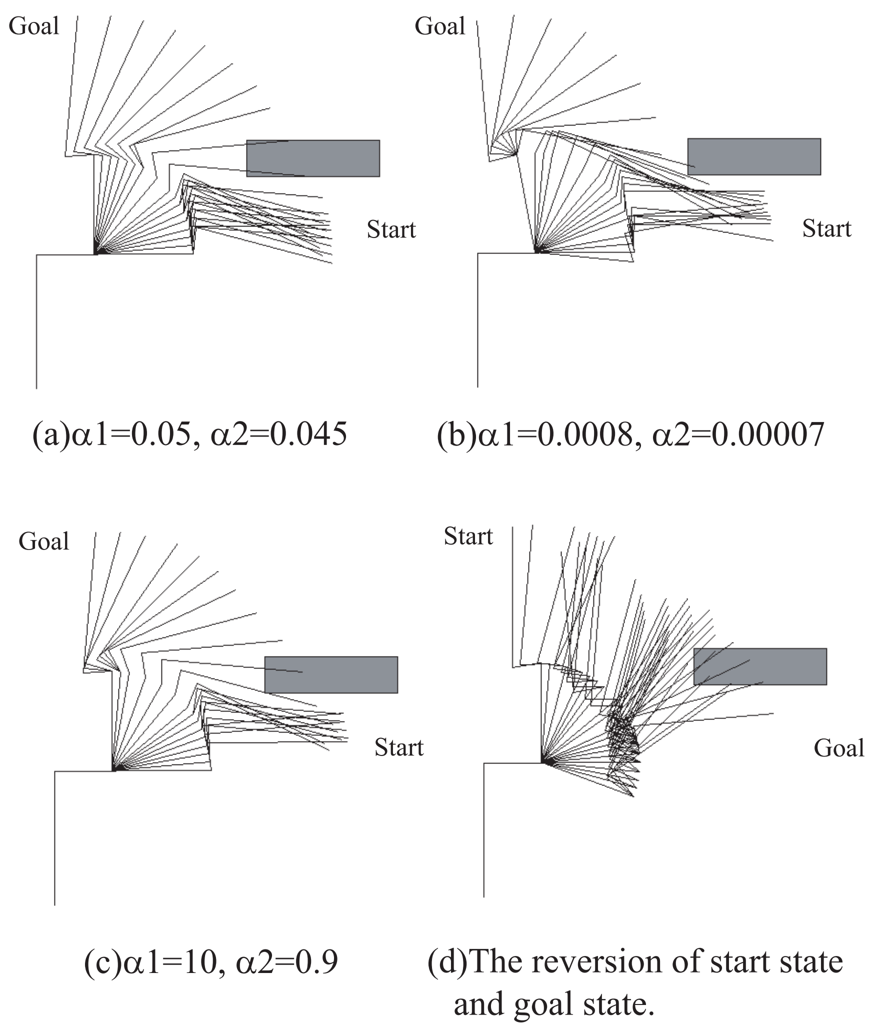

Fig. 2.11: The path planning will fail when unfit parameters are given.

2.5.4 The Case with Non-suitable α

In some cases, such as the initial state and the object state changes, it may be very difficult to find suitable α to find a right path. For example, in example 1, if the initial state and the object state are reversed, suitable $\alpha 1$ and $\alpha 2$ have not ever been found. The result is shown as (d) in Fig. 2.11.

This problem can be seen from joint space. As shown in Fig. 2.12, the path to Goal 1 can be found easily. Because in one side arms will be driven by the attraction of the goal. In other side they will be repelled by the obstacle. On the contrary, it will be very hard to find a path to Goal 2. Because in one side arms are driven by the attraction of the goal, but in other side they are rejected by the obstacle. The two strengths will clash each other. So it will be difficult to find a path in this case.

If Start and Goal 1 is reversed, it can be seen that the path planning will become difficult because the same reason.

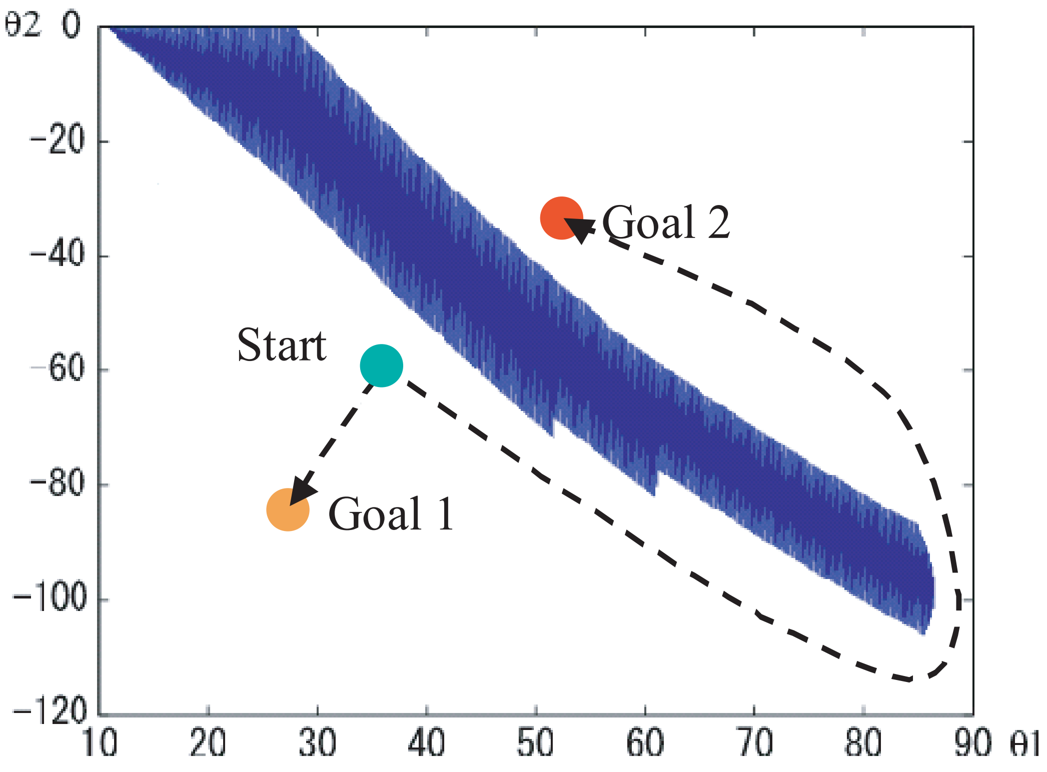

Fig. 2.12: Easy way and hard way.

2.5.5 Resolving Method

According to section 2.5.4, it is necessary to find resolving method when the right path can not be found.

Search towards More Directions

A search of 2-DOF has 4 directions before. In order to solve the problem in section 2.5.4, the search towards 8 directions is simulated. The new directions are 45, -45, 135, and -135 degrees. Fig. 2.13 shows the search.

The purpose of this simulation is to see whether there is stronger search ability by using 8 directions search. At present, it is necessary to set sub-goals in order to get right path in some cases.

Although the result is basically same with the search towards 4 directions, the 'end-effector vibration' in the simulation before is decreased. And the new method searched for 29 steps. The old method needs 33 steps.

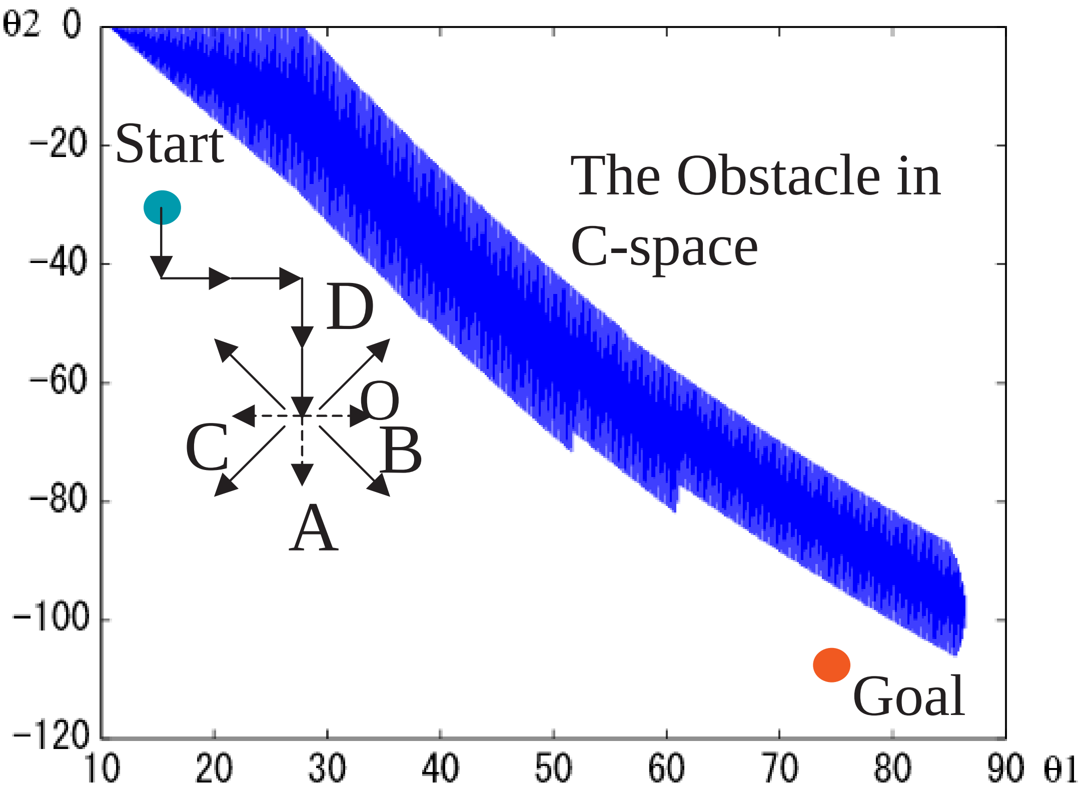

Fig. 2.13: Search towards more directions.

Using Global Planner

Global planner is a good method to improve the success rate of the search. It sets a subgoal and gives the local planner the promising direction so that the local planner can search path faster.

As it is not so obvious where the subgoal should be located, the global planner tries to generate some sub-goals. Each subgoal is tested by the local planner until the collision-free path is found.

2.6 Global Planner

The global planner sets a sub-goal and gives the local planner the promising direction so that the local planner can search path faster.

As it is not so obvious where the sub-goal should be located, the global planner tries to generate some sub-goals. Each subgoal is tested by the local planner until the collision-free path is found. In this section. we discuss how to create global planner.

The creation of global planner is difficult than I thought. Although it is very easy for human being to decide subgoals.

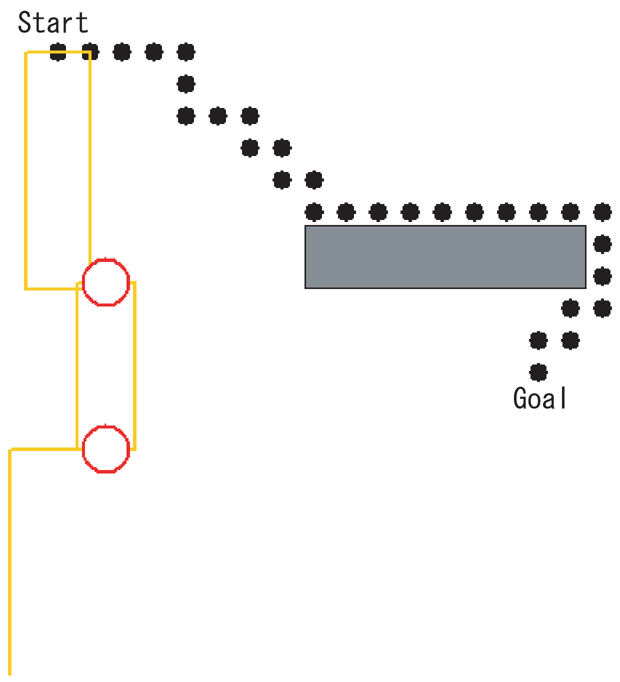

Fig. 2.14: The result of an end-effector search. The dots are path searched out.

I considered a method to make global planning. The method is to make path planning by using end-effector. The method is

1. Make path planning only for end-effector.
2. Solve the configuration of every joint by inverse kinematics according to the path of end-effector.

But it failed. Fig. 2.14 shows an example of 2-DOF robot. The start configuration is (90 degrees, 0 degrees). The goal configuration is (0 degrees, 0 degrees). We can see the robot can not go through the obstacle according to the path of end-effector. I think the reason of failure is that, if only end-effector is considered, it is not enough. There is not only end-effector for a robot.

Two methods below are stated to make global planning.

2.6.1 Method 1: Increasing Discretization Angle

Maintain the estimation function and increase discretization angle to make global planning. For example, increase discretization angle from 5 degrees to 15 degrees. It is effective by the provement of simulation.

The demerit of this method is that it is not always successful. But the method is fast.

2.6.2 Method 2: Using A* Algorithm

Using A* algorithm, global planning can be made. Discretization angle can be from 10 degrees to 20 degrees.

The demerit of this method is slow. But the method is always successful.

2.7 Simulation

In all the simulations and the experiments of this chapter, the surrounding is decided as: The size of the whole surrounding is 1.2m×1.2m×1.2m. The surrounding is divided into 5 levels by Octree. The size of smallest unit is 3.25cm×3.25cm×3.25cm.

The robot used in the experiments is FANUC Robot, LR Mate 200iB. In the simulations, the model of it is used.

2.7.1 Simulation of 2-DOF Manipulator

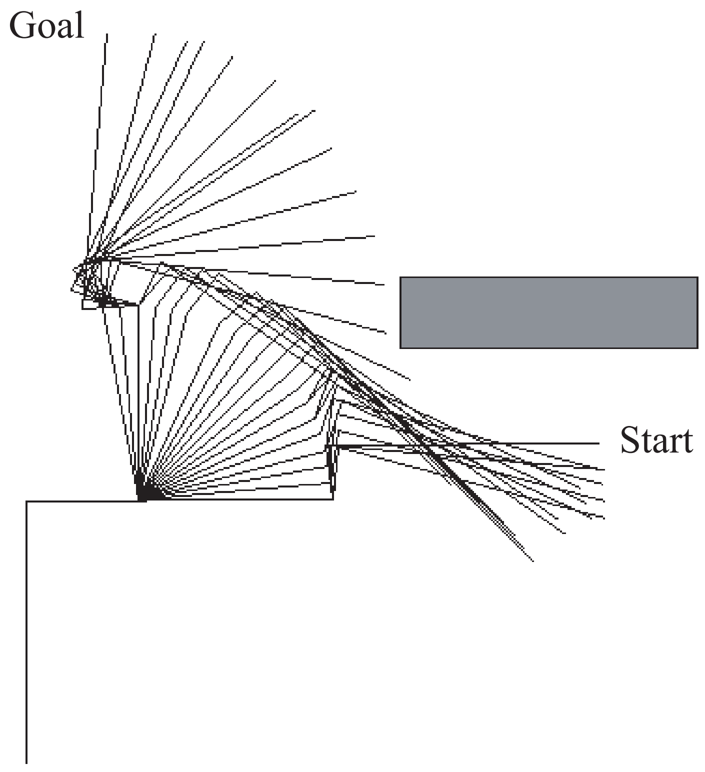

Fig. 2.15: The simulation result of 2-DOF robot manipulator.

The proposed path planning for 2-DOF robot manipulator is simulated. The model of the robot in the simulation is based on link 2 and link 3 of FANUC Robot, LR Mate 200iB. We developed a simulator for 2-DOF Manipulator using Visual C++ 6.0.

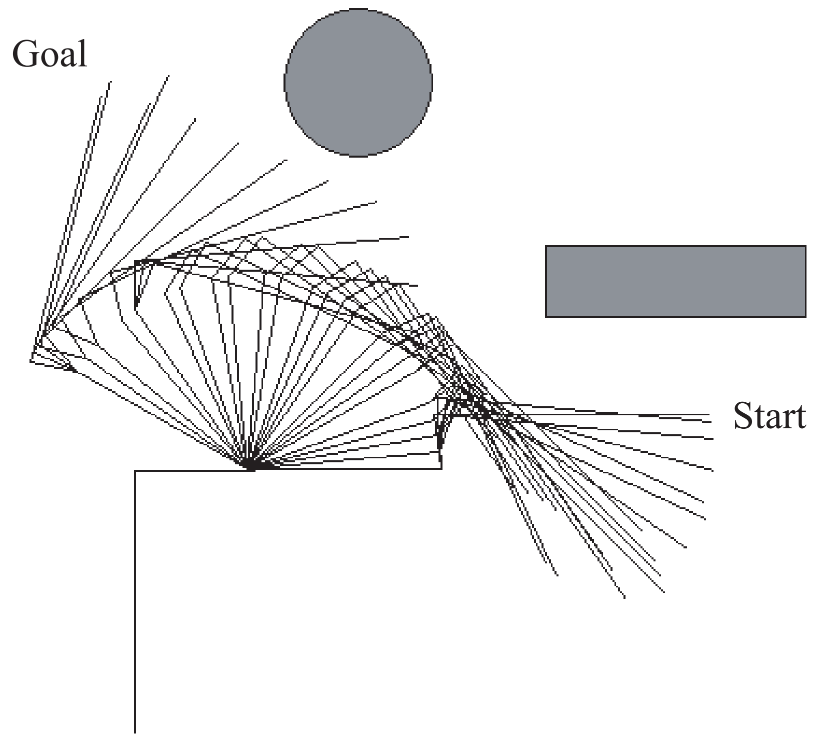

Fig. 2.16: Another simulation of 2-DOF robot manipulator. One more obstacle is added in surrounding.

In Fig. 2.15, the obstacle is a rectangle. The thickness of the rectangle is 10cm. The length of the rectangle is 45cm.

The start configuration of the robot is (0 degrees, 0 degrees). The goal configuration is (90 degrees, 0 degrees). The whole process is shown in the figure.

Another simulation is shown in Fig. 2.16. One more obstacle is added in the surrounding. Then two obstacles need to be avoided. One is a rectangle, another is a circle.

2.7.2 Simulation of 3-DOF Manipulator

The path planning for 3-DOF manipulator is simulated. In order to simulate path planning in 3-D space, we developed a 3-D simulator using DirectX 9.0.

When the simulator is made, the model of robot is created by a software named MetasequoiaLE. '.X' file is generated to describe the model of robot. Then Visual C++ 6.0 is used, DirectX programming is done.

As shown in Fig. 2.17, The obstacle is a cubic. The size of the cubic is $0.5\text{m}\times0.3\text{m}\times0.51\text{m}$.

From (1) to (6) in the figure, we can see the robot's links is going around the obstacle in order to avoid collision.

The calculation time is 0.8s with 1.0GHz CPU. The time for building Octree is contained in it. The time for building Octree is about 0.7s.

From the result, it can be seen that the time for building Octree occupies the most of the whole calculation time.

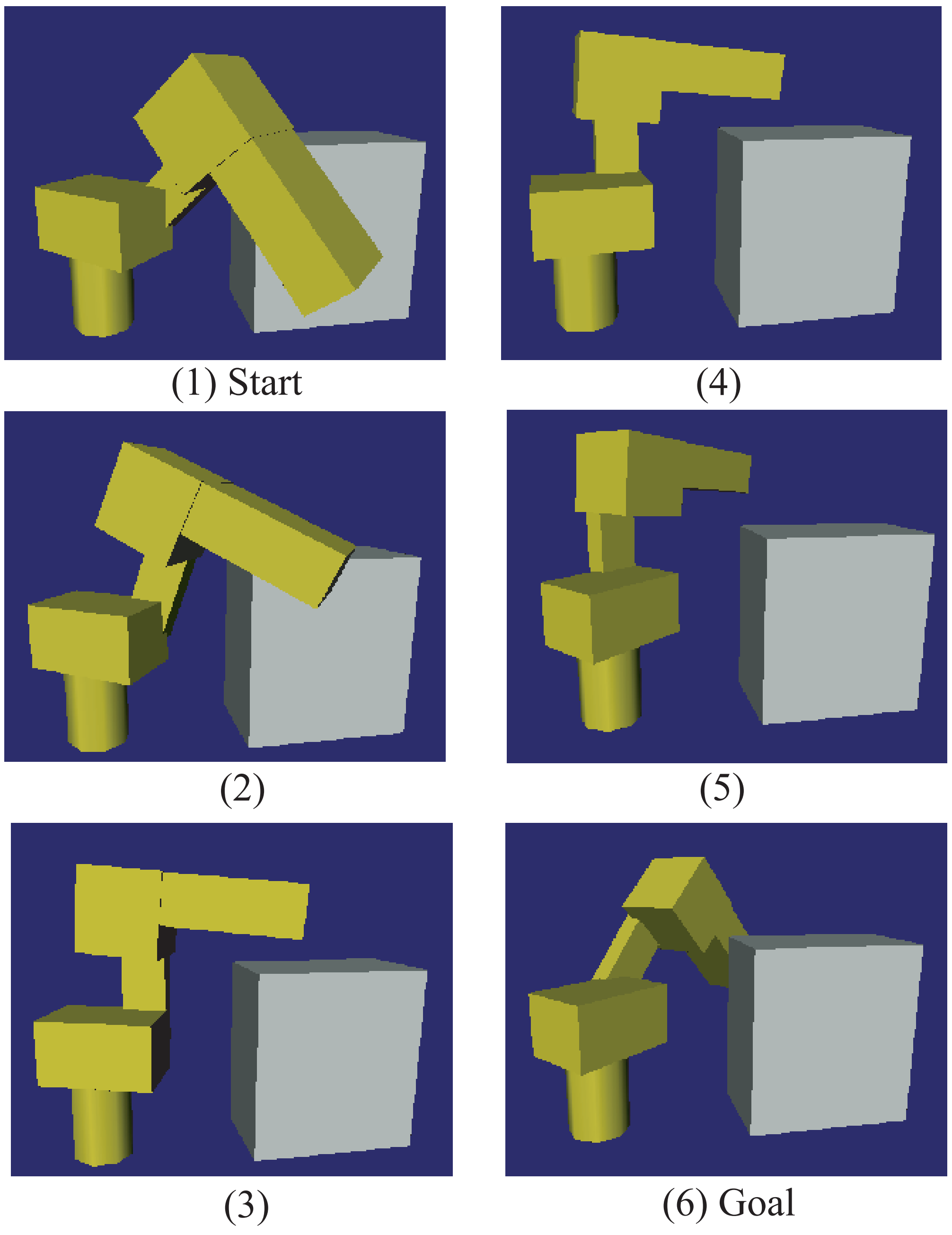

Fig. 2.17: The simulation result of 3-DOF robot manipulator.

2.7.3 Simulation of 5-DOF Manipulator

A simulator for 5-DOF Manipulator is built. As shown in Fig. 2.18, there is a cubic object and a box which is on the top of the cubic object. The cubic object is same with the object in the last simulation. The box is a cubic whose size is 0.2m×0.2m×0.2m. The thickness of its frame is 5cm.

It is shown that the manipulator is entering into the box and avoiding collision from (1) to (6) in the figure. The calculation time of path planning is 1.3s with 1.0GHz CPU. The time for building Octree is also contained in it.

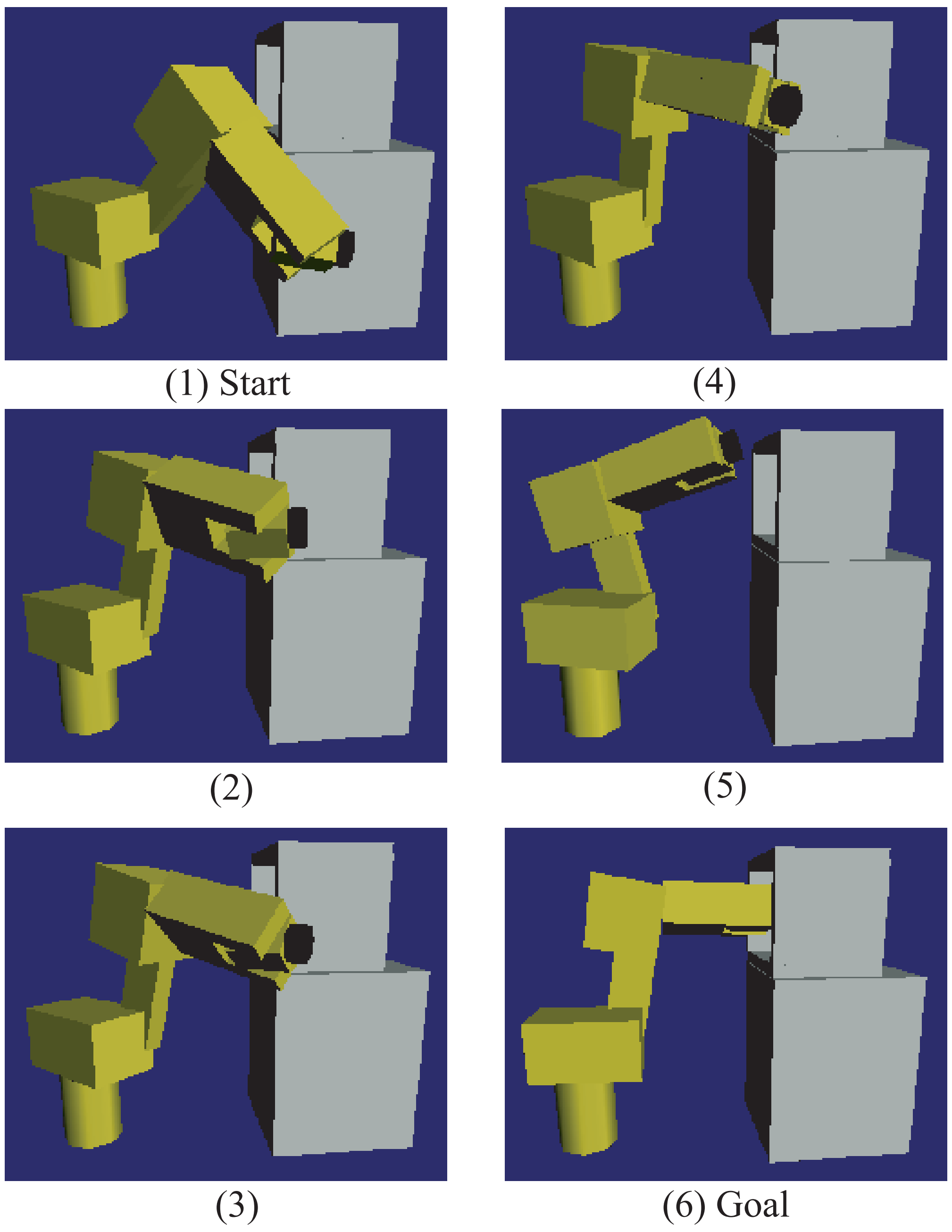

Fig. 2.18: The simulation result of 5-DOF robot manipulator.

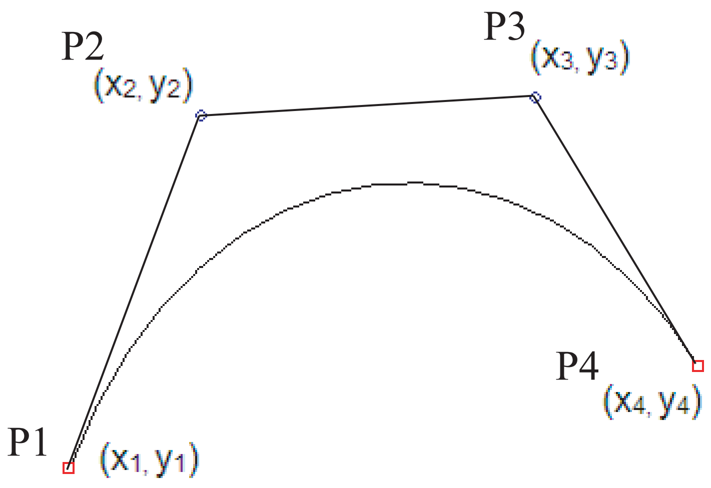

Fig. 2.19: Smooth the path by B-spline method.

2.8 Experiment

2.8.1 Trajectory Generation

Path generation only relates to obstacle avoidance, not including the velocity and acceleration of a robot. Trajectory will involve the velocity and acceleration of a robot.

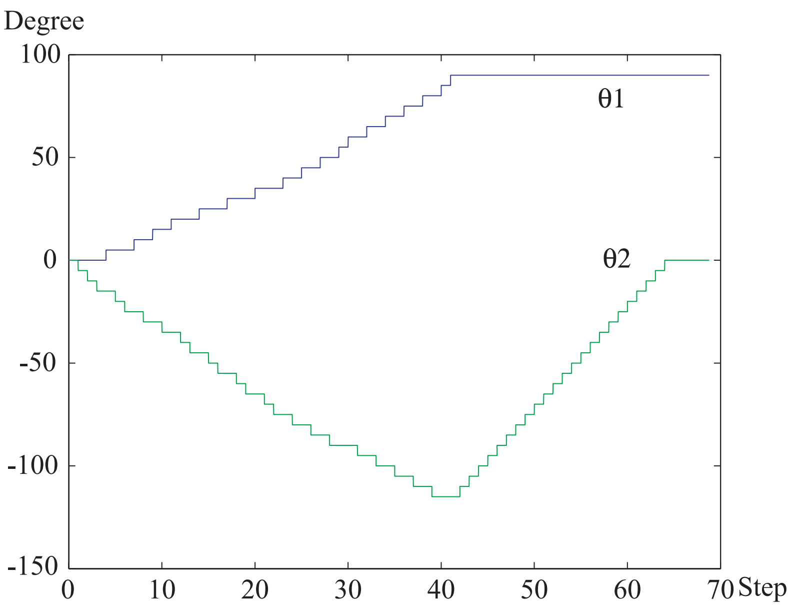

Fig. 2.20: The path generated.

The generated path is not smooth. If it was used as the instruction to a robot, the velocity and acceleration would be very big. Therefore the path must be smoothed before a real experiment.

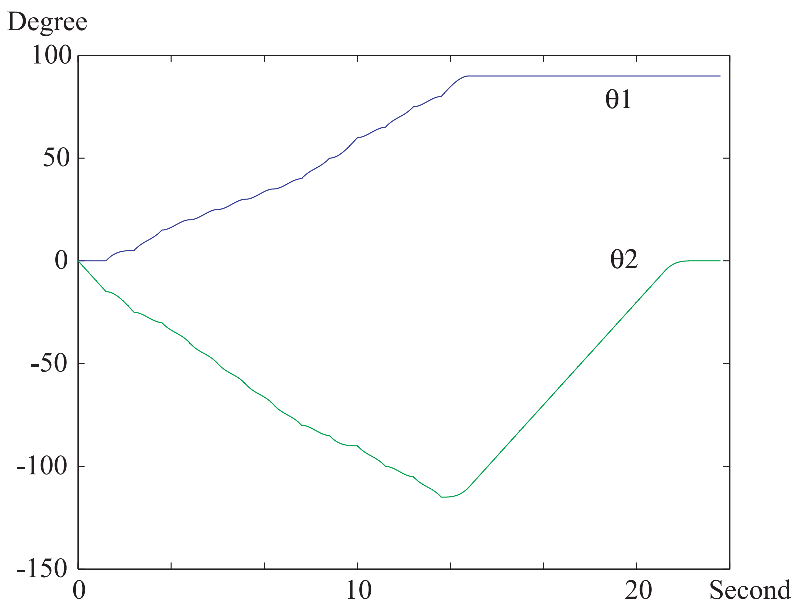

Fig. 2.21: The result after smoothing the path.

The method of B-Spline is used to smooth a curve. As shown in Fig. 2.19, the direct lines among p1, p2, p3, p4 need to be smoothed in order to generate a curve. The method of B-Spline can be used to get a curve. Equation 2.14 and 2.15 are used to calculate the coordinate of the points on curve.

$$x = (1-t)^3 x_1 + 3(t-1)^2 t x_2 + 3(1-t)t^2 x_3 + t^3 x_4 \tag{2.14}$$

$$y = (1-t)^3 y_1 + 3(t-1)^2 t y_2 + 3(1-t)t^2 y_3 + t^3 y_4 \tag{2.15}$$

In equation 2.14, 2.15, $0 \leq t \leq 1$.

The method is applied to smooth the path in the experiment. Fig. 2.20 and Fig. 2.21 separately shows a path before smoothing, and the result after smoothing. The path after smoothing will be the instruction to robot.

2.8.2 System Structure of Experiment

As shown in Fig. 2.22, the system structure of the experiment is illustrated.

In the experiment, a path is generated at first. Then the path is smoothed to get the trajectory of the robot. The trajectory instruction is sent to the robot as the reference. And the robot actuate the instruction. The robot is controlled by a PC, 700 MHz, 256 MB RAM. The program of control is C language program. The strategy of control is PID control.

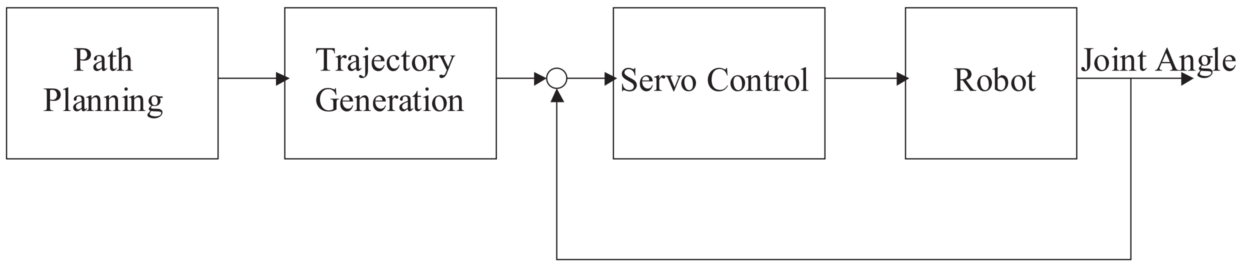

Fig. 2.22: The system structure of the experiment.

2.8.3 Experiment Result

The experiment of obstacle avoidance for 2 joints(only use joint 2 and joint 3) is realized using FANUC robot, LR Mate 200iB. The result is shown in Fig. 2.23. As shown in the figure, the rectangle on the top of the cubic object is the obstacle. Joint 2 and joint 3 of the robot is used to act as two links. From (1) to (7) in the figure, we can see the robot is avoiding the obstacle and going to the object position.

In Fig. 2.24, it is the experiment of obstacle avoidance for 3 joints (using joint1, joint 2 and joint 3). As shown in the continuous photos, the robot is going around the obstacle in order to avoid collision.

2.9 Summary

This chapter presents the approach using Octree model. In this approach, Octree model is used to express environment and detect collision. The arms of robot in space are expressed by a set of points on the surface of them. Then the distance between manipulator and obstacle is estimated by the potential generated by Octree modeling. Based on the distance estimated, graph search is executed to find a path. The effectiveness of the approach is proved by simulations and the experiments realized by a FANUC robot, LR Mate 200iB.

The merit of this approach is that it is very fast. The calculation time is less 1.5 second. It can act as a real-time approach to make path planning for robot manipulator. In the calculation time, the time for building Octree occupies a big portion.

The demerit of this approach is that it is not always successful. Therefore it is necessary to make global path planning. In this thesis, two method of global path planning is proposed. One is to increase discretization angle. Another is to use A* algorithm.

It is very easy to make global path planning by the knowledge of human being. But it is not practical to use the knowledge of human being when a path planning is made for advanced robot, such as mobile manipulator. So automatic global path planning is needed.

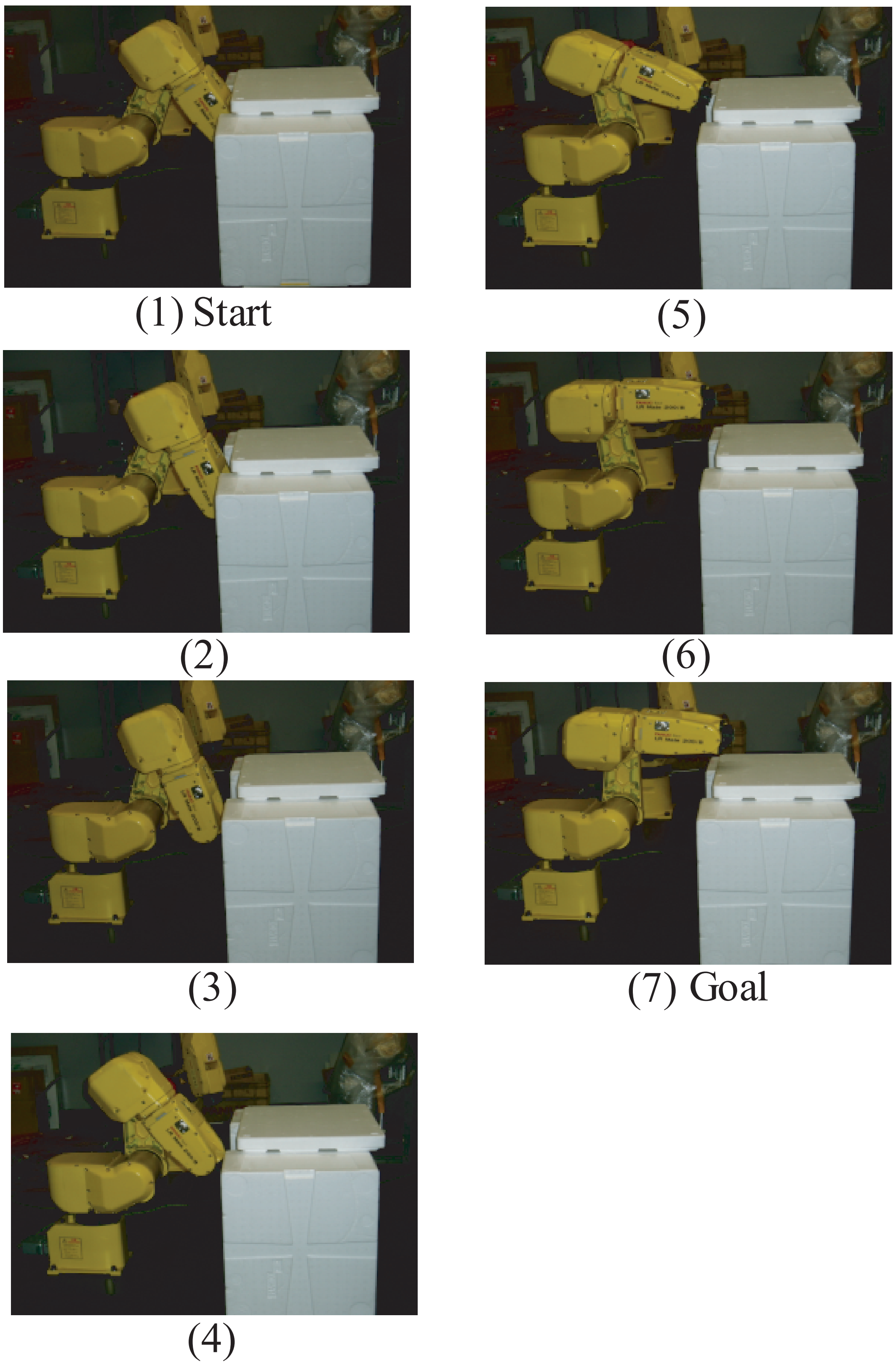

Fig. 2.23: The experiment result of 2 joints.

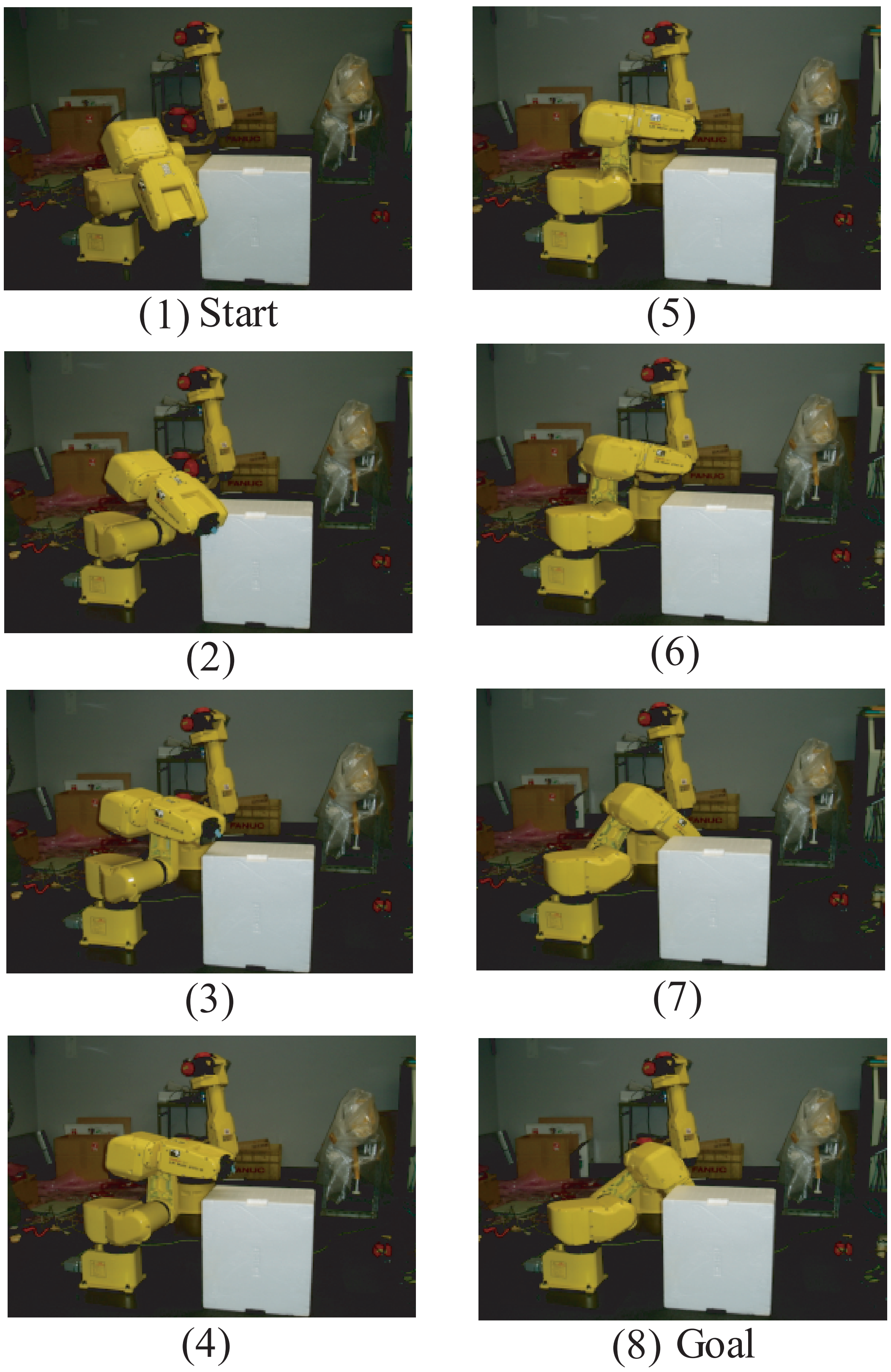

Fig. 2.24: The experiment result of 3 joints.

Chapter 3

The Graph Search Approach Using A* Algorithm

3.1 The Outline of the Approach

In order to prove the advantage of the algorithm using Octree, another approach is used to compare with the approach using Octree.

The approach is a graph search approach using A* algorithm. The approach uses the concept of global path planning and local path planning to speed up calculation. Therefore 'A* + Subgoal' is the main character of the approach. If only A* is used, the calculation time will be very long. In this approach, A* algorithm is used in both global planning and local planning.

The approach uses two strategies. One is graphic search in C-space by A* algorithm. The other is subgoal. Subgoal is created by global planning, which is a gross planning. It gives a general directions of a path. Then local planning is executed between every 2 sub-goals.

Therefore, 'A* + Subgoal' is the main character of the approach. If only A* is used, the calculation time will be very long.

A* algorithm is used in both global planning and local planning. A* algorithm is a algorithm to find a shortest path between two points. The algorithm will be described in section 3.2.

3.2 A* Algorithm

3.2.1 Introduction of A* Algorithm

A* algorithm (pronounced 'A star') was first described in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael. A* algorithm is the most popular choice for path finding, because it's fairly flexible and can be used in a wide range of contexts.

A* algorithm is a graph search algorithm that finds a path from a given initial node to a given goal node (or one passing a given goal test). It employs a 'search estimate' that ranks each node by an estimate of the best route that goes through that node. It visits the nodes in order of this search estimate. The A* algorithm is therefore an example of best-first search.

3.2.2 The Search Area

The problem of search can be seen as someone wants to get from start point to goal point. There is wall or other obstacle between the two points. Fig. 3.1 illustrates a case, with a starting point, an ending point, and an obstacle (wall) in between.

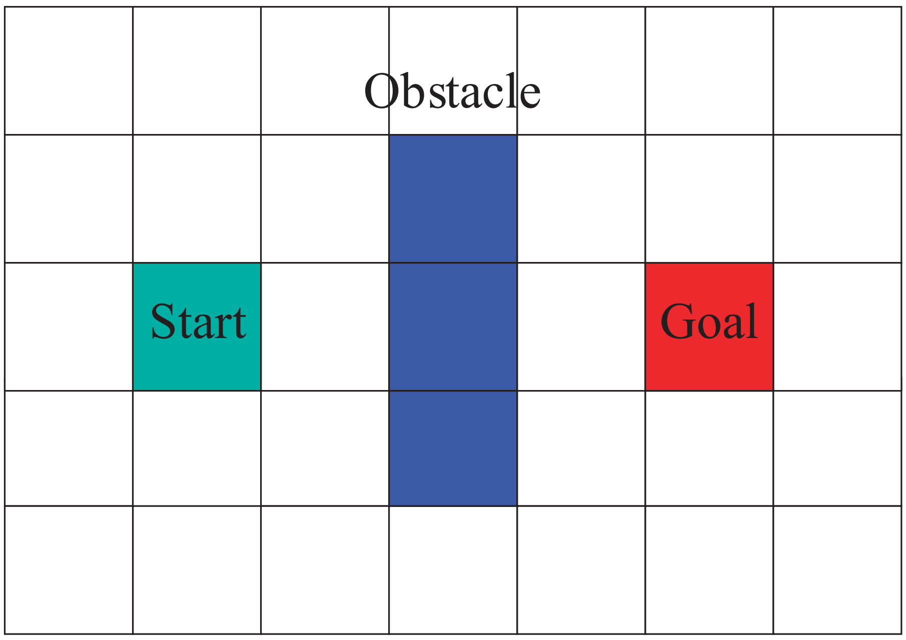

Fig. 3.1: The search area.

The first thing that should be noticed is that the search area has been divided into a square grid. Simplifying the search area, as that has been done here, is the first step in path finding. This particular method reduces the search area to a simple two dimensional array. Each item in the array represents one of the squares on the grid, and its status is recorded as walkable or unwalkable. The path is found by figuring out which squares should be taken to get from the start point to the end point. Once the path is found, the person moves from the center of one square to the center of the next until the target is reached.

These center points are called 'nodes'. Why not just refer to them as squares? Because it is possible to divide up your path finding area into something other than squares. They could be rectangular, hexagons, or any shape, really. And the nodes could be placed anywhere within the shapes, in the center or along the edges, or anywhere else. This system is used as example, however, because it is the simplest.

3.2.3 Starting Search

Once the search area has been divided into a manageable number of nodes, as the grid layout has been done with above, the next step is to conduct a search to find the shortest path. In A* path finding, it is done this by starting point, checking the adjacent squares, and generally searching outward until target is found out.

The search is begun by doing the following:

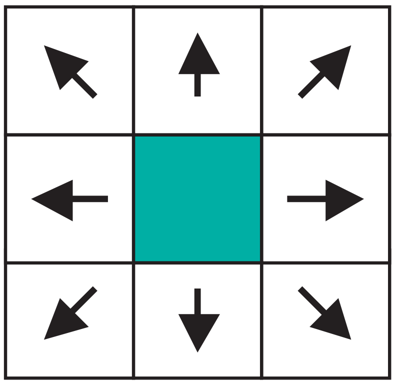

Fig. 3.2: Search directions.

1. Begin at the start point and add it to an 'open list' of squares to be considered. The open list is kind of like a shopping list. Right now there is just one item on the list, but there will be more later. It contains squares that might fall along the path you want to take, but maybe not. Basically, this is a list of squares that need to be checked out.

2. Look at all the reachable or walkable squares adjacent to the start point, ignoring squares with walls, water, or other illegal terrain. Add them to the open list, too. For each of these squares, save the start point as its 'parent square'. This parent square stuff is important when the final path will be traced by it. It will be explained more later.

3. Drop the start square from the open list, and add it to a 'closed list' of squares that you don't need to look at again for now. At this point, you should have something like the following illustration. In this diagram, the light dark square in the center is the start square. All of the adjacent squares are now on the open list of squares to be checked. Each has a gray pointer that points back to its parent, which is the starting square.

3.2.4 Path Scoring

The key to determining which squares to use when figuring out the path is the following equation:

F = G + H

where

G = the movement cost to move from the start point to a given square on the grid, following the path generated to get there.

H = the estimated movement cost to move from that given square on the grid to the final destination, the end point.

The definition of H is often referred to as the heuristic, which can be a bit confusing. The reason why it is called that is because it is a guess. The actual distance is not known until the path is found out, because all kinds of stuff can be in the way (walls, water, etc.). There are a few methods to calculate H.

The path is generated by repeatedly going through the open list and choosing the square with the lowest F score. This process will be described in more detail a bit further in the article. First look more closely at how the equation is calculated.

As described above, G is the movement cost to move from the starting point to the given square using the path generated to get there. In this example, a cost of 10 will be assigned to each horizontal or vertical square moved, and a cost of 14 for a diagonal move. Use these numbers because the actual distance to move diagonally is the square root of 2 (don't be scared), or roughly 1.414 times the cost of moving horizontally or vertically. 10 and 14 are used for simplicity's sake. The ratio is about right, and it is avoided to calculate square roots. Using whole numbers like these is a lot faster for the computer. Path finding can be slow if short cuts are not used like these.

Since the G cost is calculated along a specific path to a given square, the way to figure out the G cost of that square is to take the G cost of its parent, and then add 10 or 14 depending on whether it is diagonal or orthogonal (non-diagonal) from that parent square. The need for this method will become apparent a little further on in this example, as more than one square are gotten from the beginning of the starting square.

H can be estimated in a variety of ways. The method used here is called the Manhattan method, where you calculate the total number of squares moved horizontally and vertically to reach the target square from the current square, ignoring diagonal movement. Then multiply the total by 10. This is called the Manhattan method because it's like calculating the number of city blocks from one place to another, where you can't cut across the block diagonally. Importantly, when calculating H, any intervening obstacle is ignored. This is an estimate of the remaining distance, not the actual distance, which is why it's called the heuristic.

F is calculated by adding G and H. The results of the first step in the search can be seen in the illustration below. The F, G, and H scores are written in each square. As is indicated in the square to the immediate right of the starting square, F is printed in the top left, G is printed in the bottom left, and H is printed in the bottom right.

So look at some of these squares. In the square with the letters in it, G = 10. This is because it is just one square from the starting square in a horizontal direction. The squares immediately above, below, and to the left of the starting square all have the same G score of 10. The diagonal squares have G scores of 14.

The H scores are calculated by estimating the Manhattan distance to the target square, moving only horizontally and vertically and ignoring the wall that is in the way. Using this method, the square to the immediate right of the start is 3 squares from the target square, for a H score of 30. The square just above this square is 4 squares away (remember, only

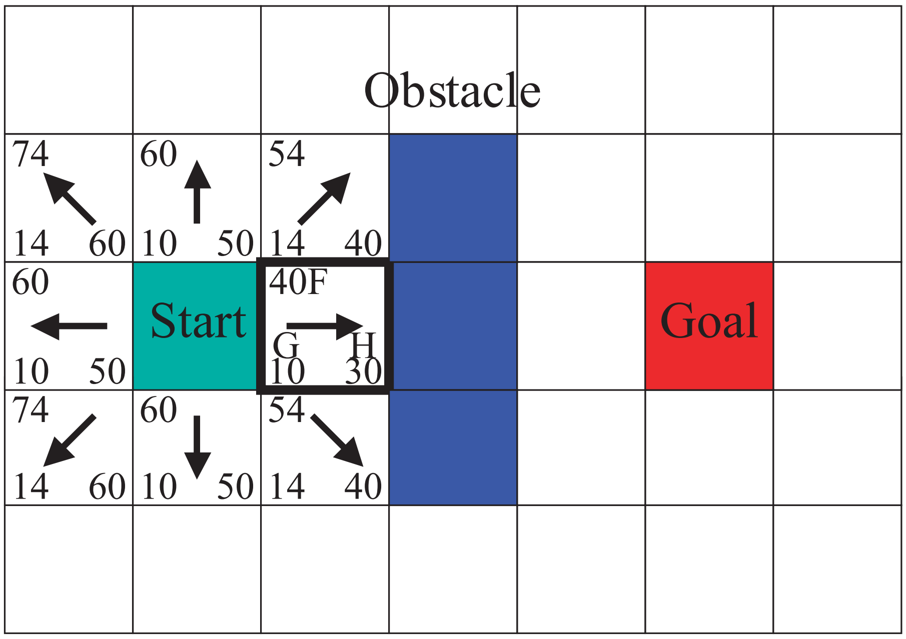

Fig. 3.3: The first step of the search.

move horizontally and vertically) for an H score of 40. It can be seen how the H scores are calculated for the other squares.

The F score for each square, again, is simply calculated by adding G and H together.

3.2.5 Continuing the Search

To continue the search, choose the lowest F score square from all those that are on the open list. Then do the following with the selected square:

4. Drop it from the open list and add it to the closed list.

5. Check all of the adjacent squares. Ignoring those that are on the closed list or unwalkable (terrain with walls, water, or other illegal terrain), add squares to the open list if they are not on the open list already. Make the selected square the 'parent' of the new squares.

6. If an adjacent square is already on the open list, check to see if this path to that square is a better one. In other words, check to see if the G score for that square is lower if the current square is used to get there. If not, don't do anything. On the other hand, if the G cost of the new path is lower, change the parent of the adjacent square to the selected square (in the diagram above, change the direction of the pointer to point at the selected square). Finally, recalculate both the F and G scores of that square. If this seems confusing, you will see it illustrated below.

In the initial 9 squares, there are 8 squares left on the open list after the starting square was switched to the closed list. Of these, the one with the lowest F cost is the one to the immediate right of the starting square, with an F score of 40. So select this square as the next square. It is with bold frame in Fig. 3.4.

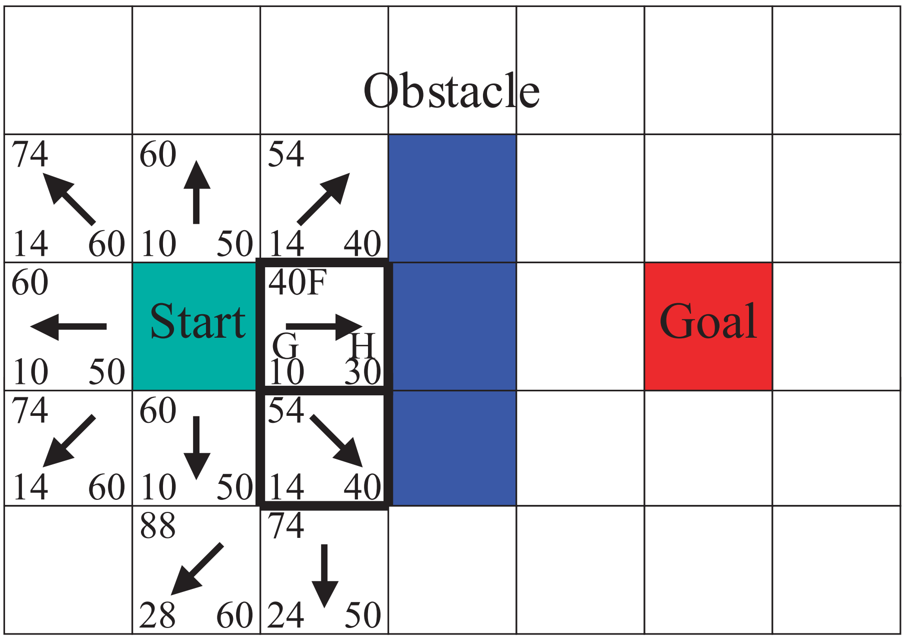

Fig. 3.4: Continue the search.

First, drop it from the open list and add it to the closed list. Then check the adjacent squares. Well, the ones to the immediate right of this square are wall squares, so we ignore those. The one to the immediate left is the starting square. That's on the closed list, so ignore that, too.

The other four squares are already on the open list, so it is needed to check if the paths to those squares are any better using this square to get there, using G scores as the point of reference. Let's look at the square right above the selected square. Its current G score is 14. If we instead went through the current square to get there, the G score would be equal to 20 (10, which is the G score to get to the current square, plus 10 more to go vertically to the one just above it). A G score of 20 is higher than 14, so this is not a better path. That should make sense if you look at the diagram. It's more direct to get to that square from the starting square by simply moving one square diagonally to get there, rather than moving horizontally one square, and then vertically one square.

When this process is repeated for all 4 of the adjacent squares already on the open list, it is found that none of the paths are improved by going through the current square, so don't change anything. So now that look at all of the adjacent squares, do with this square, and ready to move to the next square.

So go through the list of squares on the open list, which is now down to 7 squares, and pick the one with the lowest F cost. Interestingly, in this case, there are two squares with a score of 54. So which is chosen? It doesn't really matter. For the purposes of speed, it can be faster to choose the last one you added to the open list. This biases the search in favor of squares that get found later on in the search, when you have gotten closer to the target. But it doesn't really matter. (Differing treatment of ties is why two versions of A* may find different paths of equal length.)

So let's choose the one just below, and to the right of the starting square, as is shown in the following illustration.

This time, when the adjacent squares are checked, it is found that the one to the immediate right is a wall square, so ignore that. The same goes for the one just above that. The square just below the wall is also ignored. Why? Because you can't get to that square directly from the current square without cutting across the corner of the nearby wall. You really need to go down first and then move over to that square, moving around the corner in the process. (Note: This rule on cutting corners is optional. Its use depends on how the nodes are placed.)

That leaves five other squares. The other two squares below the current square aren't already on the open list, so add them and the current square becomes their parent. Of the other three squares, two are already on the closed list (the starting square, and the one just above the current square), so ignore them. And the last square, to the immediate left of the current square, is checked to see if the G score is any lower if you go through the current square to get there. No dice. So it is ready to check the next square on the open list.

This process is repeated until the target square is added to the open list, at which point it looks something like the illustration below.

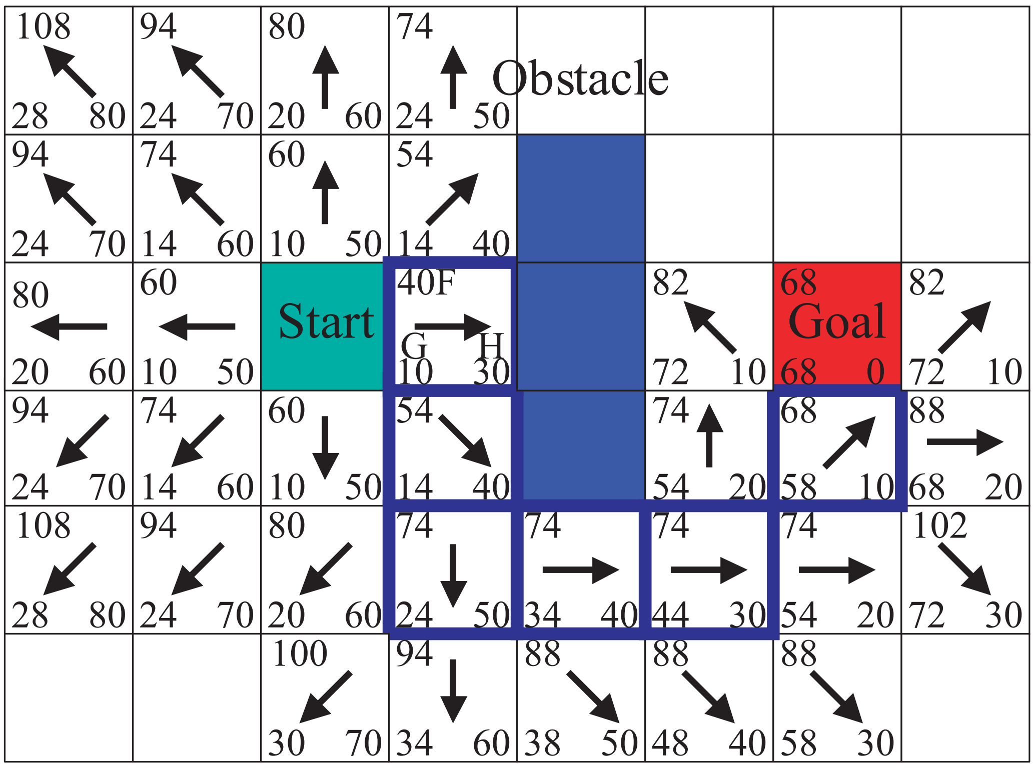

Fig. 3.5: The found path.

Note that the parent square for the square two squares below the starting square has changed from the previous illustration. Before it had a G score of 28 and pointed back to the square above it and to the right. Now it has a score of 20 and points to the square just above it. This happened somewhere along the way on the search, where the G score was checked and it turned out to be lower using a new path. So the parent was switched and the G and F scores were recalculated. While this change doesn't seem too important in this example, there are plenty of possible situations where this constant checking will make all the difference in determining the best path to your target.

So how to determine the actual path itself? Just start at the target square, and work backwards moving from one square to its parent, following the arrows. This will eventually take you back to the starting square, and that's your path. It should look like the following illustration. Moving from the starting square A to the destination square B is simply a matter of moving from the center of each square (the node) to the center of the next square on the path, until you reach the target.

3.2.6 Summary of A* Algorithm

There are 3 lists in this algorithm. Open list, closed list, and parent list. Open list contains squares that might fall along the path you want to take, but maybe not. Basically, it is a list of squares that need to be checked out. Closed list contains squares that you don 't need to look at again. The squares in parent list will decide the path found.

The basic cost function is:

$$F = G + H \tag{3.1}$$

g is the cost given in terms of the number of traced squares from the initial square to a square in the area of the grids. h is the cost of traced squares from the given square to the end square. The 'f' of smallest cost will be selected, which means the path is shortest.

The process of the A* algorithm will be:

1) Add the starting square (or node) to the open list.

2) Repeat the following:

a) Look for the lowest F cost square on the open list. Refer to this as the current square.

b) Switch it to the closed list.

c) For each of the 8 squares adjacent to this current square $\cdots$

If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.

If it isn 't on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.

If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square.

d) Stop when:

Add the target square to the closed list, in which case the path has been found (see note below), or Fail to find the target square, and the open list is empty. In this case, there is no path.

3) Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square. That is the path.

3.3 Collision Checker

The approach before uses Octree to check collision. The checker is not complete. In the new approach, there will be not Octree model to check collision. Then a normal collision checker is used. This algorithm is shown below:

At first, face and vertex need to be expressed by mathematics.

Vertex can be expressed easily by its coordinate (x,y,z).

Face can be express by an equation like equation 3.2.

$$ax + by + cz + d = 0 \tag{3.2}$$

In the equation, let (a,b,c) as an unit vector. It means $\sqrt{a^2 + b^2 + c^2} = 1$.

Then use the vector (a,b,c,d) to express a face. Name the vector as *f*.

The vector (a,b,c) is the normal vector for the face. The side of the vector is recognized as positive sphere of the face. The opposite side of the vector is recognized as negative sphere of the face. Positive sphere can be recognized as the sphere outside the space. Negative sphere can be recognized as the sphere inside the space.

We can check whether a point is inside a face or outside a face by using equation 3.3.

$$DP = a \times x + b \times y + c \times z + d \tag{3.3}$$

The equation is the dot product of vector *f* and the coordinate vector of point.

When *DP*=0, the point is on the face. When *DP*<0, the point is inside the face. When *DP*>0, the point is outside the face.

There are two cases of collision in work space. One is that face collides with vertex. Another is edge collides with edge.

3.3.1 Vertex-Face Collision

In the case of surface-vertex, equation 3.4 can be used to detect collision.

$$\begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_{nf} \end{pmatrix} \begin{pmatrix} v_1 v_2 \dots v_{nv} \end{pmatrix} = \begin{pmatrix} f_1 v_1 & f_1 v_2 & \dots & f_1 v_{nv} \\ f_2 v_1 & f_2 v_2 & \dots & f_2 v_{nv} \\ \dots & \dots & \dots & \dots \\ f_{nf} v_1 & f_{nf} v_2 & \dots & f_{nf} v_{nv} \end{pmatrix} \tag{3.4}$$

In the equation, vector *f* is the vector of face. *v* is the coordinate of point.

When the values of all the rows in the right matrix of equation 3.4 are positive, it means all the points are out of the surfaces. There is no collision. If there is a row in the matrix in which all the values are negative. It means a point is in the object. Collision has happened.

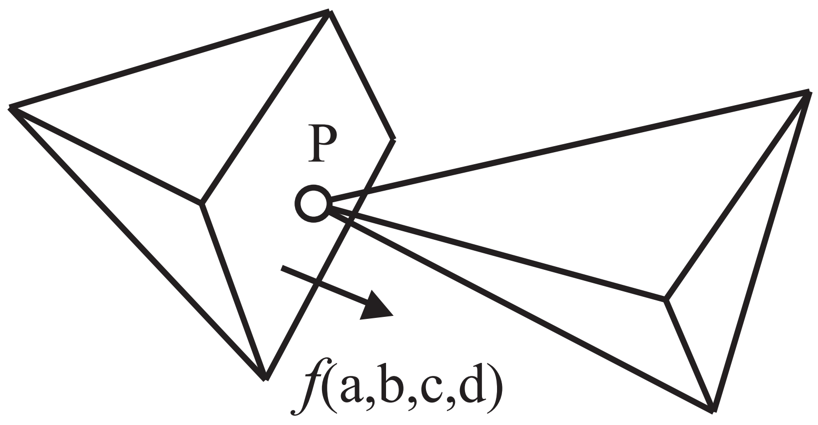

Fig. 3.6: Vertex-face collision.

3.3.2 Edge-Edge Collision

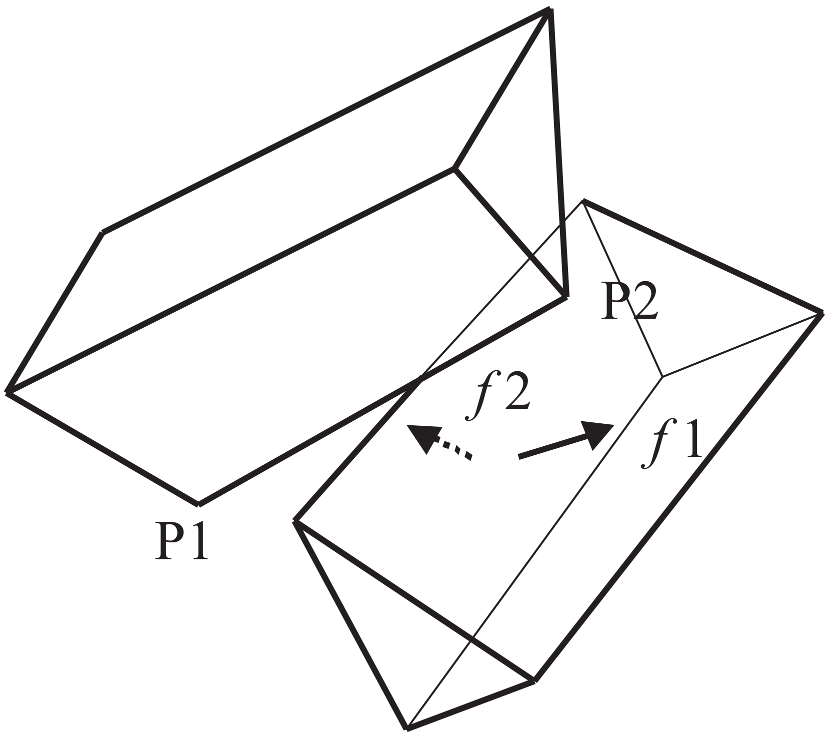

Fig. 3.7: Edge-edge collision.

In the case of edge-edge, one edge can be seen as a line between two points. Another edge can be seen as a line between two faces.

As shown in Fig. 3.7, the edge of the upper object can be a line between the points P1 and P2. The edge of the below object can be a line between the two faces, which expressed by *f*1 and *f*2.

Then equation 3.5 and 3.6 can be used to check collision. When equation 3.5 as well as

3.6 is satisfied, it is collision-free .

$$f_{i1}v_{j1} > 0, f_{i2}v_{j2} > 0 \ or \ f_{i1}v_{j2} < 0, f_{i2}v_{j1} < 0 \tag{3.5}$$

$$f_{i1}v_{j1} \times f_{i2}v_{j2} - f_{i1}v_{j2} \times f_{i2}v_{j1} > 0 \tag{3.6}$$

In the equations, the vectors f are the vectors of the faces. the vectors v are the coordinate vectors of the vertexes.

3.4 The Result of the Approach

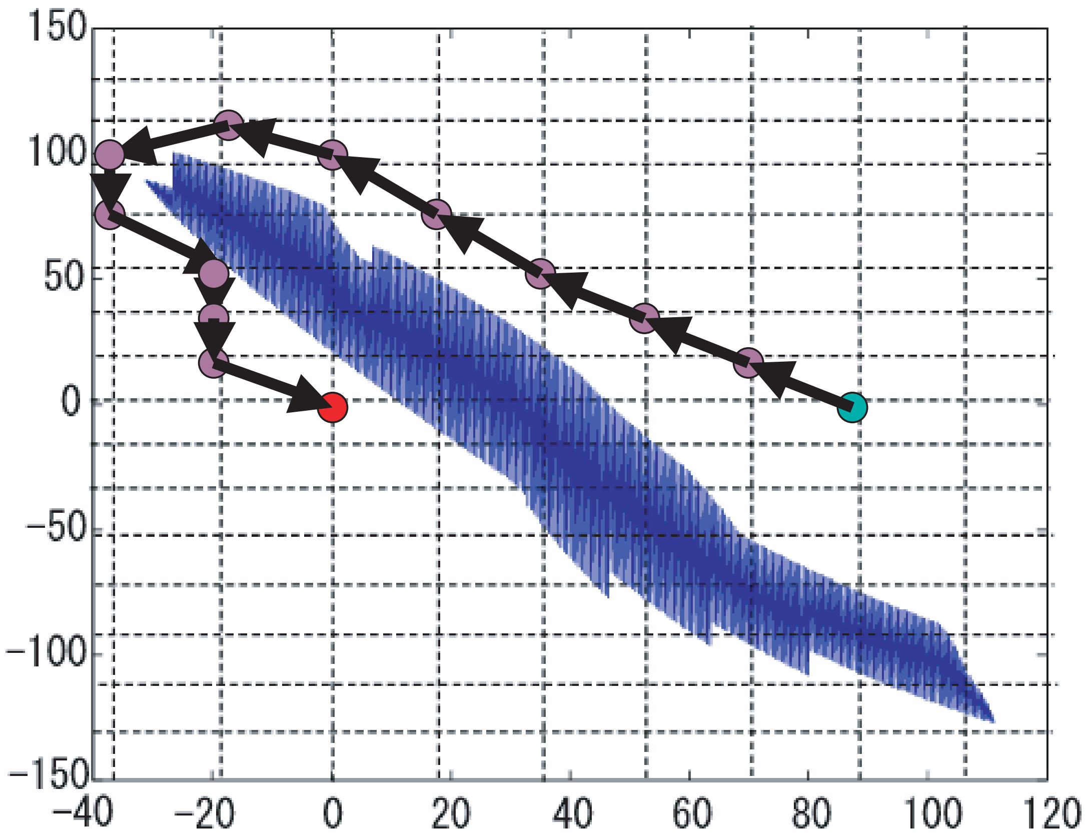

Fig. 3.8: The result of global path planning using A* algorithm is shown in C-space.

In the approach, global path planning using A* algorithm is made at first. Then local path planning using A* algorithm is executed in the subgoals which are created by the global path planning.

In Fig. 3.8 shows the result of global path planning using A* algorithm in C-space. The discretization angle is 18 Degrees. The points are subgoals created by the global path planning. The start configuration is (90 Degrees, 0 Degrees). The end configuration is (0 Degrees, 0 Degrees). The result is shown in C-space.

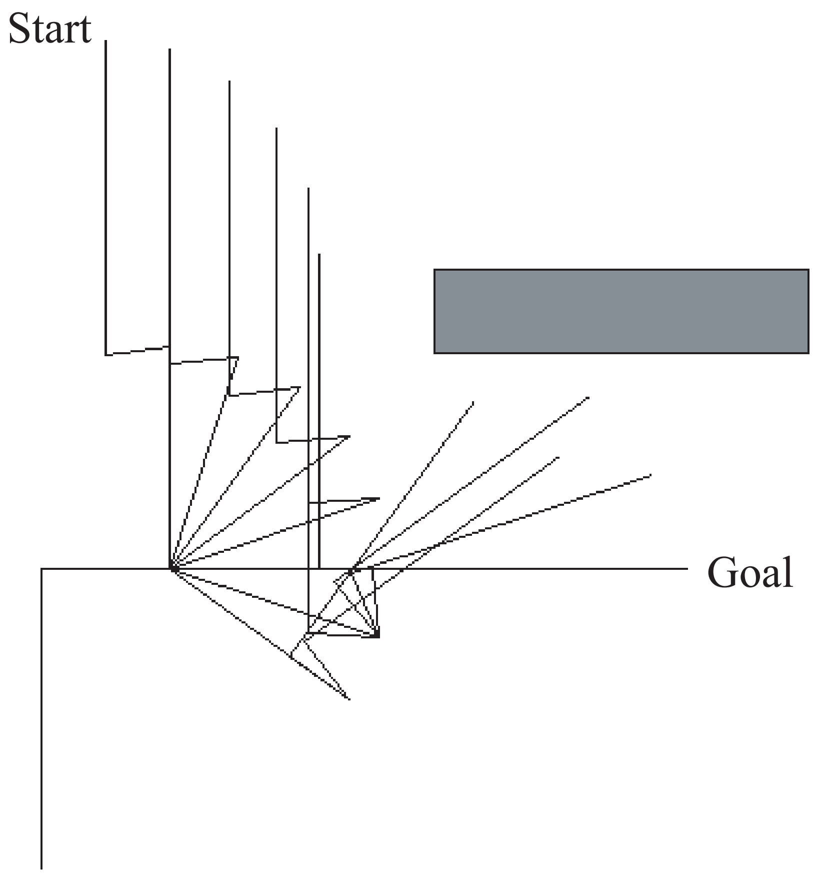

Fig. 3.9: The simulation result of global path planning using A* algorithm.

Fig. 3.9 shows the simulation result of global path planning in work space.

After global path planning, local path planning is executed. The whole process of path planning is shown in Fig. 3.10.

3.5 Compared with the Approach Using Octree

According to the result having gotten, the approach of Octree has advantage when DOF>3. When DOF<=3, the calculation time using graph search is also short. But when DOF>=5, the calculation time will increase greatly if A* algorithm is used.

The result of comparison is shown in Table 3.1. The data in the table are based on a few examples. The discretization angles for search are both 2 degrees for two approaches. The simulations is done on a same PC with 1.0GHz CPU. The result shows the approach using Octree is faster. It can act as real-time path planner. On the other hand, it may take dozens of seconds for another approach to find a path. It can hardly act as real-time path planning.

In addition, we can see graph search approach has much more collision check points. This is because there is $3^n - 1$(n is the number of DOF) possibilities for every step in graph search approach. But there is only 2×n-1 possibilities in the approach using Octree. This

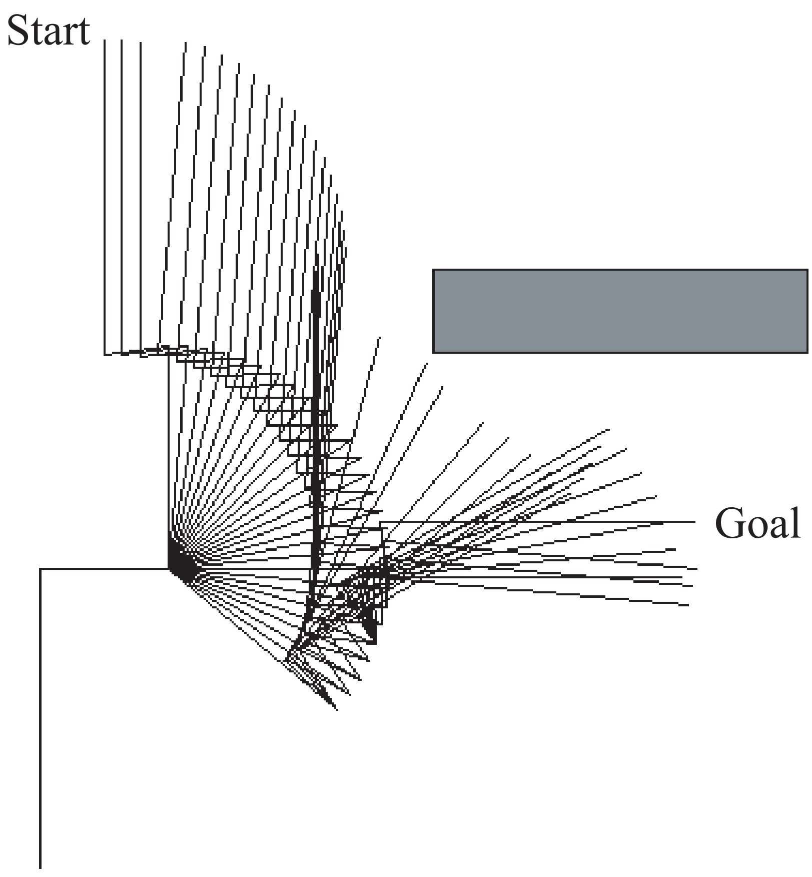

Fig. 3.10: The whole process of the simulation result using A* algorithm.

leads to the result that the approach using Octree is faster than graph search approach.

3.6 Summary

In this chapter, the approach using A* algorithm is described. The approach has two level path planning, global path planning and local path planning. Global path planning is made at first to find some subgoals. Then local path planning is made among the subgoals.

The collision checker used in this approach is different from the approach using Octree approach. The collision checked is complete. However, it is complex.

Table 3.1: Comparison of two approaches for a 5-DOF Manipulator.

Data / Approaches	Calculation Time	Collision Check Points
Octree Approach	<2 Second	10^3
Graph Search Approach	3-40 Seconds	10^4 - 10^5

The approach is simulated. Its effectiveness is proved.

The result shows the approach using Octree is faster. It can act as real-time path planner. On the other hand, it may take dozens of seconds for the approach using A* algorithm to find a path. It can hardly act as real-time path planning.

In addition, the approach using A* algorithm has much more collision check points. This is because there is $3^n - 1$(n is the number of DOF) possibilities for every step in the approach. But there is only 2×n-1 possibilities in the approach using Octree. And this leads to the result that the approach using Octree is faster than graph search approach.

Chapter 4

Conclusions

4.1 Summary

This thesis proposes a real-time approach of path planning for robot manipulator.

Path planning of a manipulator refers to a short and collision-free path finding from start position to goal position. Path planning for robot manipulator is an important research because every robotics system with manipulator requires the path planning.

A lot of research has been devoted to the issue. However, few algorithm of path planning is applicated in practical industry because this problem becomes complex and time-consuming as a many-DOF robot manipulator executes task in cluttered environments. In factory, collision-free trajectory for industrial robot is usually done by human. There are few practical path planners that is simple enough and quick enough.

On the other hand, many of future robot tasks, such as assembly and disassembly, tele-operation, and medical surgery, will be executed in dynamic environments. Therefore, a real-time path planner for many-DOF manipulator is necessary. A robot with manipulator can not react to dynamic changes in environment if it has not the capability of real-time path planning.

For the above reasons, the thesis proposes a real-time approach to generate path for robot manipulator.

The proposed approach has composite character. Generally, the approach in this research has three main features. One is using Octree model. The second one is artificial potential. The third one is searching to find a path. The approach is mainly stated as bellow.

Octree model – Octree is used to transform the information in surrounding into computer. Octree divides space into subspaces smaller and smaller. A computer will store the data of every subspace in surrounding. Based on Octree model, surrounding space is expressed in computer.

In addition, the distance between manipulator and obstacle is estimated by the way of generating artificial potential. When the value of artificial potential is big, the distance between manipulator and obstacle is short. When the value of artificial potential is small, the distance between manipulator and obstacle is long.

Moreover, collision detection is executed by Octree Model. The points on manipulator are checked whether there is obstacle in the highest level of Octree model. If there is obstacle, collision happens.

Artificial potential – Artificial potential is like electrical potential or gravity potential, but it is an artificial potential. Robot will be repelled by obstacle, and be attracted by goal after the potential is generated.

Based on Octree model, artificial potential can be generated. The points on the surface of manipulator are taken. Using kinematics, the positions of the points are calculated. Then check whether there is obstacle in the space of this level. The checks are done from the lowest level until the highest level. If there is obstacle in the space of this level, potential is generated.

Search – Path planning is executed by search step by step. Which step will be taken is decided by the action of artificial potential.

In every step, there are (2×n-1) kinds of possibility. n is DOF of manipulator. The evaluation function will be calculated separately for every kind of possibility. The best step will be selected by the value of the evaluation function.

The effectiveness of the approach is proved by the simulations and the experiments.

2-DOF and many-DOF simulator is developed separately. The many-DOF simulator is developed using DirectX by Visual C++ 6.0.

The experiment is realized by a FANUC robot, LR Mate 200iB. In experiment, the path must be smoothed at first. Because the path generated is not smooth. If it was used as the instruction to a robot, the velocity and acceleration of robot would be very big.

The method of B-Spline is used to smooth path. The path after smoothing will be the instruction to robot.

At last, the calculation time of proposed approach is compared with the result of the typical graph search approach.

The typical approach uses two strategies. One is graph search in C-space by A* algorithm. The other is the strategy of global planning and local planning. In this strategy, subgoal is created by global planning, which is a gross planning. It gives a general directions of a path. Then local planning is executed between every 2 subgoals. In addition, a normal collision checker is used in this approach.

The calculation result shows that the approach using Octree is faster.

4.2 Future Work

In the future, there are two plans in the area. They are:

1. Vision-based path planning.
2. The path planning in a dynamic surrounding.

4.2.1 Vision-Based Path Planning

A very important step towards autonomous robotics is developing ways to generate path plan for achieving certain goals while satisfying environmental constraints. Classical path planning is defined on a configuration space which is assumed to be known, implying the complete knowledge of both the robot kinematics as well as knowledge of the obstacles in the configuration space. On the other hand, vision-based, or more general, sensor-based path planning provides a more practical approach to robot control. To best utilize the sensor feedback, a robot motion plan should incorporate constraints from the sensor system as well as criteria for optimizing the quality of the sensor feedback.

4.2.2 The Path Planning in a Dynamic Surrounding

The dynamic planning is a path planning in dynamic environment. In the research of this thesis, all the obstacles are static. However, there are movable obstacles in surrounding. The path planning in such a surrounding is dynamic planning.

In the dynamic planning, dynamic constraints such as bounded acceleration are necessary to be studied. And in dynamically changing environment it is very difficult to know where obstacle is. The method to obtain the environment information should be researched.

List of Publications

1. Lu Wu and Yoichi Hori, Real-time Collision-free Path Planning for Robot Manipulator based on Octree Model, Proc. Of the 9th. IEEE Conference of AMC-2006, 2006.3.27-29, Istanbul.

2. 呉 魯, 堀 洋一, オクトツリーを用いたロボットマニピュレータのリアルタイム障害物回避軌道計画, 電気学会産業計測制御研究会, IIC-06-30, 2006.3.8-9。

Biblography

[1] Yong K. Hwang and Narendra Ahuja, "Gross motion planning - a survey", ACM Computing Surveys (CSUR) archive Vol 24, Issue 3, pp. 219 - 291, Sep. 1992.

[2] K. Hamada and Y. Hori, "Octree-based approach to real-time collision-free path planning for robot manipulator" , AMC '96-MIE. IEEE vol.2, 1996 4th International Workshop on Robotics and Automation, pp. 705 - 710, Mar. 1996.

[3] Tsutomu Hasegawa, "Collision Avoidance of a 6DOF Manipulator Based on Empty Space Analysis of the 3-D Real World", Proc. of IEEE IROS'90, pp. 583 - 589, 1990.

[4] P.C. Chen and Y.K. Hwang, "SANDROS: a motion planner with performance proportional to task difficulty", Proceedings of IEEE International Conference on Robotics and Automation, vol.3, pp. 2346 - 2353, May 1992.

[5] Shingo Ando, "A fast collision-free path planning method for a general robot manipulator", Proceedings. ICRA '03. IEEE International Conference on Robotics and Automation, Vol, 2, pp. 2871 - 2877, Sep. 2003

[6] B. Faverjon, "Obstacle avoidance using an octree in the Cspace of a manipulator", Proceedings of IEEE International Conference on Robotics and Automation, Vol. 1, pp504 - 512, Mar. 1984

[7] K. Kondo, "Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration ", IEEE Transactions on Robotics and Automation, Vol. 7, pp. 267 - 277, Jun. 1991

[8] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots", Proceedings of IEEE International Conference on Robotics and Automation, Vol. 2, pp. 500 - 505, Mar. 1985

[9] N. Okada, Minamoto, K. and Kondo, E., "Collision avoidance for a visual-motor system with a redundant manipulator using a self-organizing visual-motor map ", Proceedings of the IEEE International Symposium on Assembly and Task Planning, pp. 104 - 109, May 2001

[10] K. Watanabe, and K. Izumi, "A survey of robotic control systems constructed by using evolutionary computations" , 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on 'Systems, Man, and Cybernetics', Vol. 2, pp. 758 - 763, Oct. 1999.

[11] B. Faverjon,; P. Tournassoud, "A local based approach for path planning of manipulators with a high number of degrees of freedom"; Robotics and Automation. Proceedings. 1987 IEEE International Conference on Volume 4, Mar 1987 Page(s):1152 - 1159.

[12] C. Helguera, S. Zeghloul, "A local-based method for manipulators path planning in heavy cluttered environments"; Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on Volume 4, 24-28 April 2000 Page(s):3467-3472 vol.4 Digital Object Identifier 10.1109/ROBOT.2000.845266.

[13] http://www.ke.ics.saitama-u.ac.jp/kondo/Geomap/GeomapProgSpecJ/Chap03/Chap0303.html

[14] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee. Robotics : Control, Sensing, Vision, and Intelligence. McGraw-Hill, 1987.

[15] 浜田広一, 修士論文：多自由度ロボットの高速障害物回避軌道計画法

[16] Richad P. Paul, 吉川恒夫訳, "ロボットマニピュレータ", コロナ社,1984.

[17] 小林尚登等, "ロボット制御の実際", コロナ社,1997.

[18] 美多勉, 大須賀公一, ロボット制御工学入門, コロナ社,1989.

AppendixA

Inverse-Kinematics of Robot Manipulator

A method of inverse kinematics is presented here. The top view and side view of the first 3 manipulator's links(The manipulator is 6 DOF. The first 3 links (link 1, link 2 and link 3.) are shown in Fig. A.1. Point P is the end-effector of the first 3 joints.

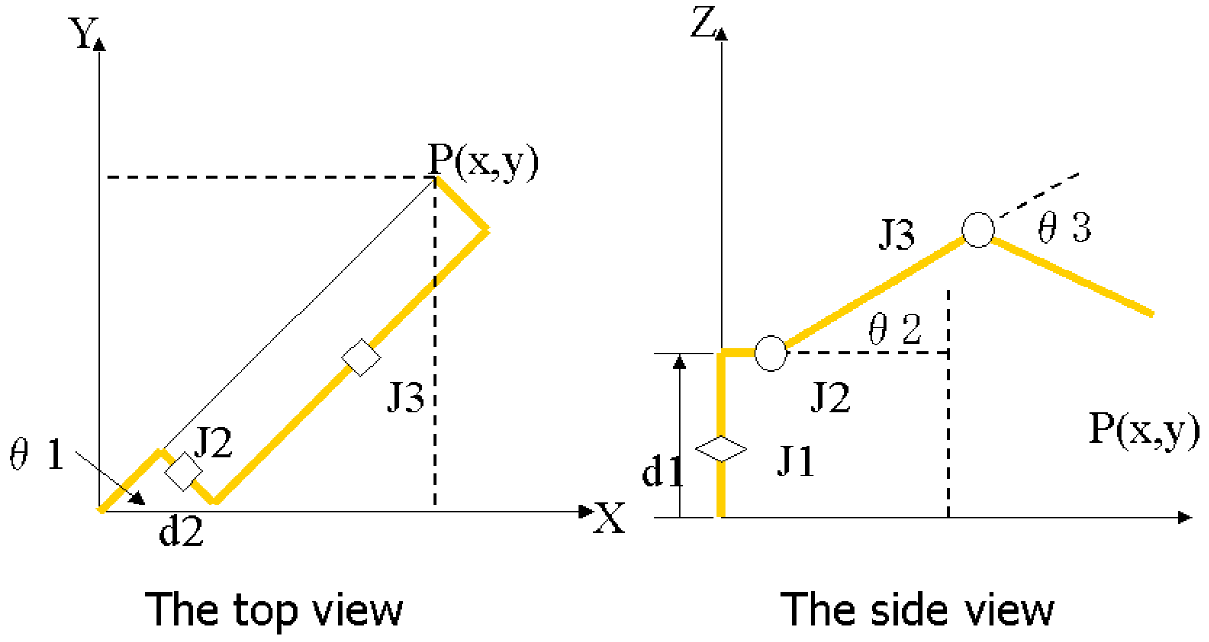

Fig. A.1: A top view and a side view of the first 3 manipulator's links. **J1, J2, J3** are the joints of robot. θ**1,** θ**2,** θ**3** is the joint angles.

$\theta 1$, $\theta 2$, $\theta 3$ can be calculated by geometric method by the coordinate of the point P. Then using the transformation matrix ${}^{i-1}A_i$, $\theta 4$, $\theta 5$, $\theta 6$ can be calculated sequentially.[17]

AppendixB

Manipulator Dynamics

The dynamics of manipulator can be calculated based the view of energy.[16] Any robot's link can be found to be

$$F_i = \sum_{j=1}^{6} D_{ij}\ddot{q}_j + I_{ai}\ddot{q}_i + \sum_{j=1}^{6}\sum_{k=1}^{6} D_{ijk}\dot{q}_j\dot{q}_k + D_i \tag{B.1}$$

In this equation:
F_i is the force of the ith joint.
$D_{ij} = \sum_{p=maxi,j}^{6} Trace(\frac{\partial T_p}{\partial q_j} J_p \frac{\partial T_p}{\partial q_i}^T)$
$D_{ijk} = \sum_{p=maxi,j,k}^{6} Trace(\frac{\partial T_p^2}{\partial q_j \partial q_k} J_p \frac{\partial T_p}{\partial q_i}^T)$
$D_i = \sum_{p=i}^{6} -m_p g^T \frac{\partial T_p}{\partial q_j}^p \mathbf{r}_p$
I_{ai} is the initial of the ith link's actuator.
q_i is the joint angle of the ith link.
T is 4× 4 transformation matrix.
j_p is the inertia matrix.

For the all links of manipulator, equation B.1 can be expressed by matrix simply as follows:

$$F = B(q)\ddot{q} + c(q, \dot{q}) + g(q) \tag{B.2}$$

where:
B is n× n the Inertial Matrix.
c is n× 1 Coriolis vector.
g is n× 1 Gravity vector.

AppendixC

The Setup of the Robot

The robot, Fanuc Robot LR Mate 200iB, was setup. Here the setup is introduced in brief. Fig. C.1 shows the construction of the robot system. As shown in the figure, a PC acts as a controller to control the robot. (The controller in control unit is not used.) The digital servo adapter of Fanuc Robot LR Mate 200iB acts as the interface between the PC and the servo amplifier. Optic cable is used for the communication among them. The control cycle can be set as 1ms/2ms/4ms/8ms. The shortest control cycle is 1ms.

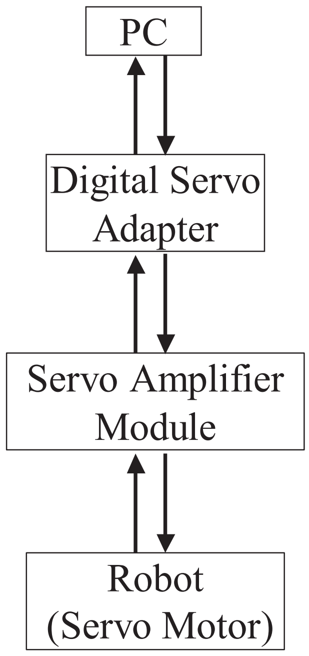

Fig. C.1: The construction of the robot system.

Fig. C.2 shows the digital servo adapter of Fanuc Robot LR Mate 200iB. As shown in the figure, Interface COP7 is used to connect with the PC. Interface COP10A is used to connect with the servo amplifier. Interface JD40 is used to connect DPL, which can write and read the parameters of the robot system.

In Fig. C.3, the control unit of Fanuc Robot LR Mate 200iB is shown. There are 4 modules in the unit. They are power supply module, 2 servo amplifier modules, and control module from left to right. The control module is not used.

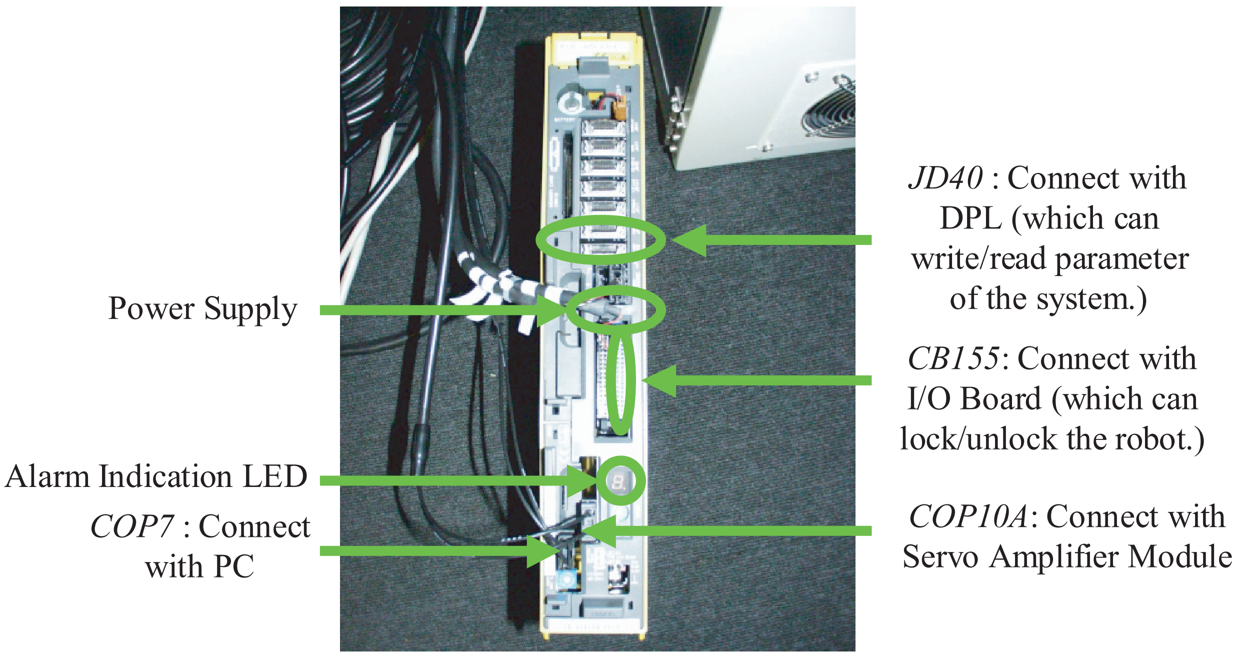
JD40 : Connect with DPL (which can write/read parameter of the system.)
Power Supply
CB155: Connect with I/O Board (which can lock/unlock the robot.)
Alarm Indication LED
COP7 : Connect with PC
COP10A: Connect with Servo Amplifier Module

Fig. C.2: The digital servo adapter of Fanuc Robot LR Mate 200iB.

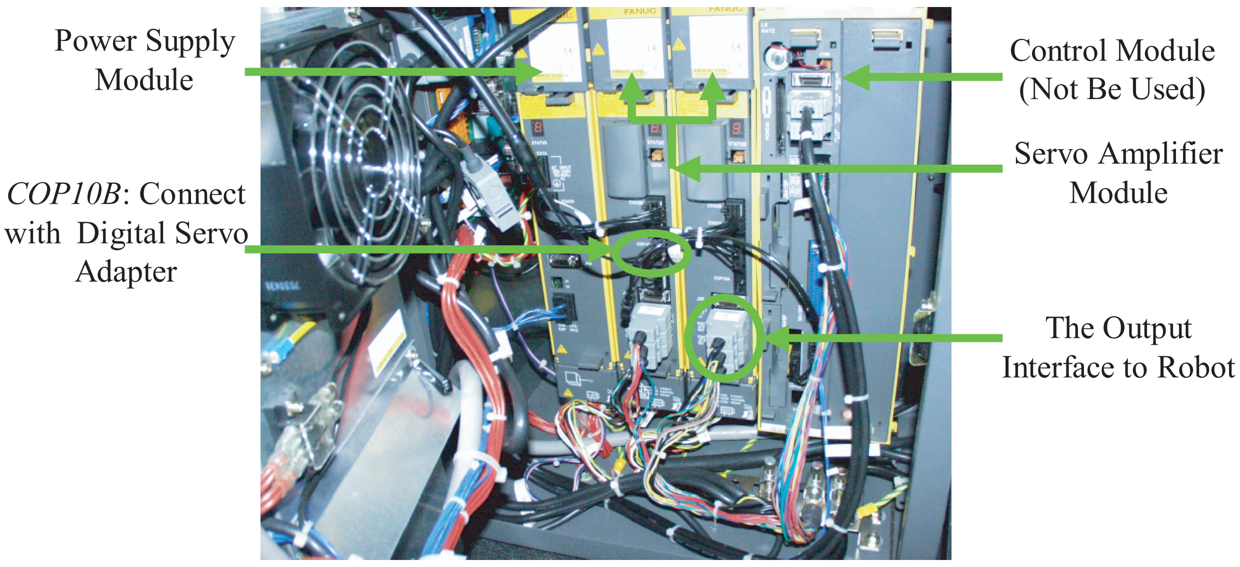
Power Supply Module
Control Module (Not Be Used)
Servo Amplifier Module
COP10B: Connect with Digital Servo Adapter
The Output Interface to Robot

Fig. C.3: The control unit of Fanuc Robot LR Mate 200iB.