A GENERAL ROBOT PATH VERFICATION SIMULATION SYSTEM

GRPVSS

by

Chungwuu Li

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Industrial Engineering and Operations Research

APPROVED:

Dr. Ching-Cheng Wang, Chairperson

Dr. Suhbash C. Sarin

Dr. John F. Jansen

August 24, 1987

Blacksburg, Virginia

A GENERAL ROBOT PATH VERFICATION SIMULATION SYSTEM

GRPVSS

by

Chungwuu Li

Dr. Ching-Cheng Wang, Chairperson

Industrial Engineering and Operations Research

(ABSTRACT)

Collision-detection is a critical task for off-line robot path planning. A general robot path verification simulation system, GRPVSS, applicable for all industrial robots with open-looped links, is created to verify the intended robot path. The manipulator and obstacles are modelled by convex polyhedra to reduce the computation burden required by the collision detection algorithm. As a kinematic simulator, GRPVSS employs motion-time profiles or ideal trapezoid profiles which describe the position-vs-time relation of an individual joint, to generate the robot working trajectory. This approach makes the to-be-verified working path closer to the real one. Both point-to-point(PTP) and continuous path(CP) operations can be simulated by GRPVSS.

Collision detection is conducted by performing geometric interference detection between the static configurations of the expanded moving robot and the static obstacles at each simulation step. In this case, the resolution of a simulation is critical to path verification. Simulations with low resolution take the risk of undetected collisions, while simulations with high resolution consume too much computing time. GRPVSS computes and employs the lowest resolution level that yields 100% path verification for the specified tolerances of manipulator dimensions. The tolerance value is specified by the user but should not be specified smaller than the positioning accuracy of the simulated industrial robot. The links of the manipulator are expanded by the amount of tolerance.

GRPVSS is a graphic simulator. A systematic control supervisor is constructed for the simulator to request input and to proceed all functions interactively with users. The robot motion of a sim-

ulated path is animated on a 3-D graphical screen. All collision configurations and related information of the simulated path are stored in a file and shown on the screen. The graphical display works on graPHIGS, one of the 3-D graphical software packages published by IBM.

Acknowledgements

I am deeply grateful to Dr. Subhash C. Sarin and Dr. John F. Jansen for being on my committee and their invaluable guidance. Special thanks go to my advisor, Dr. Ching-Cheng Wang. His help, invaluable advice, dedication, and training throughout the course of my studying in the U.S.A. are greatly appreciated.

Gratitude is also expressed to Dr. Arvid Myklebust and the CAD/CAM laboratory in Mechanical Engineering Department. Their supports in graphics facilities make this work completed. I would also like to thank for his timely discussion in the graphics package, graPHIGS.

Finally, I am indebted to my parents and family for their continuous love and support in everything I choose to do.

Table of Contents

Chapter 1 INTRODUCTION 1

1.1 General background of robotics 2

1.1.1 Basic structure of robots 2

1.1.1.1 Manipulator-drive assembly 3

1.1.1.1.1 Manipulator kinematics 4

1.1.1.1.2 Manipulator dynamics 7

1.1.1.2 Robot control system 7

1.1.2 Planning the robot motion 10

1.1.2.1 Trajectory planning 10

1.1.2.2 Task planning 18

1.2 Thesis Outline 22

Chapter 2 LITERATURE REVIEW 24

2.1 Robot off-line path planning 25

2.1.1 World modelling 25

2.1.1.1 Geometric primitive models 26

2.1.1.2 Characteristics of world information 26

2.1.2 Path generation 27
2.1.3 Path verification 29
2.2 Robot simulation for collision-free path verification 29
2.2.1 Necessary Characteristics of simulation for collision-free path verification 30
2.2.1.1 Trajectory tracing 30
2.2.1.2 100% collision detection guarantee 31
2.2.2 Robot simulators 32
2.2.3 Algorithms for trajectory tracing and collision detection 34
2.2.3.1 Algorithms for trajectory tracing 34
2.2.3.2 Collision detection algorithms 45

Chapter 3 RESEARCH OBJECTIVE and METHODOLOGY 50
3.1 Objective Statement 50
3.2 Methodology 51
3.2.1 World modelling 53
3.2.2 Path input 56
3.2.3 Trajectory generation mechanism 57
3.2.3.1 Trajectory generation for PTP operations 57
3.2.3.2 Trajectory generation for CP operations 60
3.2.4 Collision detection 61
3.2.5 Graphic Animation 66

Chapter 4 The STRUCTURE and ALGORITHMS of GRPVSS 67
4.1 The structure of GRPVSS 67
4.2 Algorithms of GRPVSS 71
4.2.1 Robot expansion algorithm 72
4.2.2 Intermediate configuration generation algorithms 78
4.2.2.1 Intermediate configuration generation algorithm for PTP operations 78

4.2.2.2 Intermediate configuration generation algorithms for CP operations 80
4.2.3 Interference detection 89

Chapter 5 CONCLUSION and DISCUSSION 95
5.1 Conclusion 95
5.2 Discussion 99
5.2.1 Computation complexity of major algorithms 100
5.2.2 Tolerance value 102
5.2.3 Further Extension 102
5.2.4 Application of GRPVSS 103

REFERENCE 105

Appendix A. USER'S GUIDE for GRPVSS 113
A.1 Preparation of input 114
A.1.1 Input files for world modelling 114
A.1.1.1 Input files for robot modelling 115
A.1.1.2 Input file for environment modelling 118
A.1.2 Specifications of user-supplied subroutines 119
A.1.3 Input files for path input 122
A.2 Running procedures 125
A.2.1 Initiation 126
A.2.2 Running Sample 128
A.3 Output format 137
A.4 Error messages 141
References 142

Vita 143

List of Illustrations

Figure 1. Basic control block diagram for robot manipulator(Adopted from Fu, Gonzalez, and Lee's book[22]. 9

Figure 2. Ideal trajectory of an individual joint under PTP command. 13

Figure 3. Ideal trajectory of an individual joint under PTP command(short move). 15

Figure 4. The contribution of the joint velocity of component j to the velocity vector. 39

Figure 5. Dynamic simulator block diagram(adopted from Walker and Orin's paper[76]). . . 42

Figure 6. The hierarchical structure of GRPVSS 52

Figure 7. The primitive model: a typical convex polyhedron. 55

Figure 8. The structure of intermediate configuration generating for PTP operations. 59

Figure 9. The relationship of intrusions of an obstacle and moving object. 63

Figure 10. Main flowchart of GRPVSS (Part A). 68

Figure 11. Main flowchart of GRPVSS (Part B). 69

Figure 12. The structure of robot expansion algorithm. 73

Figure 13. The parallel line of one edge of a convex polyhedron. 74

Figure 14. The mid-point of shortest distance. 76

Figure 15. The flowchart of resolution determining algorithm for PTP type operations 81

Figure 16. The flowchart of intermediate configuration generation for CP type operations . . 82

Figure 17. Multiple configurations of an imaginary robot reaching the same position and orienatation of end effector. 86

Figure 18. Four conditions of discussing the locations which have maximum displacements on the robot links. 88

Figure 19. The flowchart of interference detection algorithm. 91

Figure 20. Determining whether or not a point is inside a surface by using vector cross product. 93

List of Tables

Table 1. Execution time of four different methods of a five second trajectory(adapted from Walker and Orin's paper[76]) 43

Table 2. The calculation numbers of major algorithms of GRPVSS 100

Table 3. The execution time of major algorithms of GRPVSS(on IBM 4341) 101

Table 4. Checklist for GRPVSS 127

Chapter 1 INTRODUCTION

Robots are widely used in industry. They play a significant role in the automation. However, robotics is still far away from being mature and is one of the most promising research areas in recent years. Robots are used as a programmable spatial positioning device in most of the industrial applications. Thus, the capability of precisely and dexterously performing various designated motions is essential. On the other hand, the motion planning that considers robot capability while working out the appropriate manipulator trajectory for an assigned robot task is as important. On-line manual teaching is the most widely used technique for task planning nowadays. Unfortunately, on-line teaching is very expensive because it causes the equipments to be tied up during the planning stage. Therefore, off-line task planning has been proposed to prevent the equipments from being tied up and to reduce the production cost. The off-line task planning works out the manipulator path without running the real system. It would be very dangerous to apply the off-line prepared path on the real system if the path is not verified to be collision-free. This thesis presents a General Robot Path Verification Simulation System, GRPVSS, which is developed to do the collision-free-verification for the off-line generated paths. This simulation system can be used for all industrial robots with open-loop links of which only simplified shape descriptions are required.

The general background of robotics is introduced in section 1.1 of this chapter. This section briefly discusses the basic structure of an industrial robot and techniques explored for planning the robot trajectory. The research direction and the scope of this thesis are roughly stated in Section 1.2.

1.1 General background of robotics

The general background necessary for this thesis is introduced in this section. First, the basic structure of an industrial robot is briefly described in section 1.1.1, which has two major function components: the manipulator-drive assembly and the controller. The knowledge of the kinematics and dynamics of the manipulator-drive assembly as well as the function of the controller are fundamental to assigning appropriate trajectories for the employed robot. Then, how to plan the robot motion to perform the assigned task is introduced in Section 1.1.2. The controller controls the manipulator trying to realize the assigned work trajectory.

1.1.1 Basic structure of robots

Almost any device capable of being used for positioning objects has at some point been called a robot[10]. For example, the teleoperator used in the nuclear power plant, fixed stop positioning devices used in industrial pick-and-place tasks, and reprogrammable industrial robots used for welding, painting and assembly are all called robots. Definitions for industrial robots are more restrictive and consistent. The Robotics International Division of the Society of Manufacturing Engineers (RI/SME) defines a robot as "a reprogrammable, multi-function manipulator designed to move material, parts, tools, or specialized devices through variable programmed motions for the performance of a variety of tasks." According to the British Robot Association(BRA), a robot is "a programmable device designed to both manipulate and transport parts, tools or specialized

manufacturing implements through variable programmed motions for the performance of specific manufacturing tasks." In general, a modern robot system consists of the following two major parts:

1. the manipulator-drive assembly, which is the mechanical arm that consists of articulated links and actuators.
2. the controller, which controls the manipulator-drive assembly trying to go through the desired trajectory.

The manipulator-drive assembly and its kinematic and dynamic systems are discussed in Section 1.1.1.1. Robot control systems are introduced in Section 1.1.1.2.

1.1.1.1 Manipulator-drive assembly

The manipulator-drive assembly of a robot is made of articulated mechanical links powered by motors at the joints. There are two types of joints: the revolute joint, at which the motion between two links is rotary, and the prismatic joint, at which the motion is linear. Motion of the manipulator is achieved by joint movement driven by the motors. The joint values and their first and second derivatives are called positions, velocities, and accelerations respectively, irrespective of whether they are angular or linear. The tool used to perform the required task is mounted at the end of the robot manipulator and is called the end effector. In general, each joint provides one degree of freedom for the motion of the robot's end effector, and an n-degree-of-freedom manipulator contains at least n joints. It needs six degrees of freedom to have the tool reach an arbitrary point with an arbitrary orientation within its workspace; three degrees of freedom for position, and the rest three degrees of freedom for orientation. However, only four or five degrees of freedom are needed for some applications. In the following two subsections, the background of manipulator kinematics and dynamics necessary for this research are introduced in the stated order.

1.1.1.1.1 Manipulator kinematics

There are two coordinate systems commonly employed to describe the robot positioning and robot motion, i.e. the world coordinate system(WCS) and the joint coordinate system(JCS). The robot trajectory is the time sequence of the manipulator's intermediate configurations between the starting point and destination of a motion segment. It can be described in the end effector's coordinates w.r.t. the WCS or the manipulator's joint values. The space curve traced by the end effector is called the trajectory path, or simply, path. A human operator finds it convenient to work with the WCS. However, the robot completes its assigned job through joint motion. Kinematic analysis provides tools, i.e. direct kinematics and inverse kinematics, to interpret kinematic description between the WCS and the JCS. The direct kinematics determines the end effector's trajectory described w.r.t. the WCS for the given joint positions and joint velocities, whereas the inverse kinematics calculates the required joint positions and their time derivatives to produce the desired trajectory of end effector described w.r.t. the WCS. They are discussed separately below.

Direct kinematics

There are two alternative methods to derive the manipulator kinematics. One is the geometry-based method, and the other is the homogeneous transformation method[60,61]. Geometric technique utilizes the geometric relationship between coordinate systems to generate the mathematical transformation equations. No systematic approach is applied and the mathematical complexity depends on the kinematic structure of an individual manipulator. The complexity of the solution increases with the number of links. This method is good for robots with a small number of links.

The homogeneous transformation method is a general method in which the complexity of the solution is independent of the number of links or joints. The algorithm consists mainly of matrix multiplication; therefore it is easy to be implemented. The solution contains, in addition to the position and orientation of the end effector, also the position and orientation of any point on the

links, a feature which is especially important when the dynamics is of concern. Therefore, this method is very useful.

The Denavit and Hartenburg (DH) convention gives a systematic method for establishing homogeneous transformation matrices of two adjacent links of a mechanism[16]. The DH convention is mainly implemented in robot manipulators which consist of an open kinematic chain in which each joint contains one degree of freedom and the joint is either revolute or prismatic. With the DH convention, the basic skeleton of a robot is fixed by specifying four parameters for each link and its associated joint: θ, α, a, and d. θ is the varying coordinate of a revolute joint and is fixed for a prismatic joint. d is the varying coordinate of a prismatic joint and is fixed for a revolute joint. a and α are respectively the length and twist angle of the link. Each link has a coordinate frame(of the JCS) attached to it. In addition, the WCS is attached to the base of a robot. The coordinate of a point described w.r.t. any local coordinate frame can be transformed into the coordinate of that point described w.r.t. another local coordinate frame by a transformation matrix:

$$\underset{\sim}{p}_j = \mathbf{A}_i^j \, \underset{\sim}{p}_i \qquad (1-1)$$

where

$\underset{\sim}{p}_i$ = position vector of a point described w.r.t. frame i which is attached to link i

$\underset{\sim}{p}_j$ = position vector of the same point described w.r.t. frame j which is attached to link j

A_i^j = transformation matrix from frame i to frame j

A typical [A] matrix and its algebra are shown as below:

$$\mathbf{A}_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (1-2)$$

When j = 0 is set for Eq (1-1), the transformation matrix from frame i to the WCS is obtained, and

$$T_i^0 = A_1^0 \, A_2^1 \cdots A_i^{i-1} = \begin{bmatrix} u_x & v_x & w_x & p_x \\ u_y & v_y & w_y & p_y \\ u_z & v_z & w_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad (1-3)$$

where

$[u_x, u_y, u_z]^t$ is the X-axis of frame i represented in world coordinates.

$[v_x, v_y, v_z]^t$ is the Y-axis of frame i represented in world coordinates.

$[w_x, w_y, w_z]^t$ is the Z-axis of frame i represented in world coordinates.

$[p_x, p_y, p_z]^t$ is the position of the origin of frame i represented in world coordinates.

Inverse Kinematics

The inverse kinematics solves for the joint values and their time derivative of a given end effector's trajectory described w.r.t. the WCS. The existence of the closed-form solution of the inverse kinematics depends on the manipulator structure. Unlike the direct kinematics, there is no systematic method which can be employed to derive the closed form solution of the inverse kinematic problems of all manipulators with open-looped links. In fact, for some manipulator designs, the efforts for deriving closed form solutions of the inverse kinematics have been failed. However, for other designs of special kinematic structure, closed form solutions of inverse kinematic do exist. Pieper[62] developed a method to solve the close-form inverse kinematics solutions for a manipulator with six degrees of freedom in which three consecutive axes intersect at a point. Because of the difficulty of closed form solution, numerical methods are proposed to solve the inverse kinematics problems of general structures[56,72,74]. Due to their iterative nature, numerical methods require much more computations. Even worse, their convergence to the correct solutions is not guaranteed. Thus, manipulator designers are forced to design manipulators that have closed form inverse kinematic solutions. The difficulty of solving the inverse kinematic problem has had practical effects in manipulator design, many industrial robots are designed with three intersected consecutive axes in order to apply Pieper's method or with three parallel consecutive axes in order to use the geometric methods.

1.1.1.1.2 Manipulator dynamics

Kinematic analysis does not consider the force effect. Dynamics properly refers to the forces and torques acting on the joints of a manipulator that correspond to a particular kinematic state(position, velocity, and acceleration). More explicitly, dynamic analysis relates the forces/torques at joints with the end effector's trajectory. The manipulator dynamics can be divided into two categories: the inverse dynamics and the direct dynamics. The inverse dynamics computes the torque to be applied at the joints in order to achieve the desired trajectory, and the direct dynamics computes the end effector's trajectory given the torques applied at the joints.

Direct dynamics is important in its use in dynamic simulation. Dynamic simulation generates the forces/torques applied on joints according to the given control law. Then, direct dynamics gives the trajectory of the end effector according to the forces/torques applied at joints. A major role for the inverse dynamics is in generation of nominal joint torques to drive a manipulator along some specified trajectory. Since previous formulations of manipulator dynamics have been too computationally time consuming for real time control, control systems were designed by using simplified manipulator dynamics and feedback controllers. Recently, Luh et al.[50] reduced the computation complexity by using recursive dynamic form on Newton-Euler dynamic equation. Thus the computation time for more precise real-time control has been dramatically reduced.

1.1.1.2 Robot control system

By the inverse dynamics, the nominal torques applied on each joint to achieve the desired motion for the robot model can be computed. A sequence of desired positions and/or forces of the robot linkage at each time from initiation to the termination of a motion segment can be converted to a desired sequence of torque and force signal for each joint actuator. However, simply applying the pre-computed force/torque sequence on the robot would certainly fail to produce the desired

motions and forces on end effector because of the cumulative effects of unpredictable disturbances arising from modelling inaccuracies, limitations of computational accuracy, and mechanical effects such as friction or vibration. The remedy is to measure the actual motion during execution of the preplanned command sequence, to compute an error from the desired motion and forces, and to modify sequence input so as to reduce the error at subsequent times. Thus feedback control is to ensure that a planned sequence of motions and forces will execute correctly in the face of unpredictable errors. In this section, robot control system is briefly introduced.

The robot control issue is treated as a path-trajectory tracing problem, and the control system structure can be described by the basic control block diagram for robot manipulator as shown in Figure 1 in the following page. Under this consideration, motion control can be classified into three major categories[22].

1. Joint motion control: In this categories, the robot motion is controlled in the joint coordinate system(JCS) to follow a joint-interpolated trajectory. The following control schemes can be included in this category: Joint Sevomechanism(For example, the controllers of Puma 560 Series belong to this type.), Computed torque technique[42,51], Minimum time control[4,33], Variable structure control [30,75,82], Non-linear decoupled control[20,21,23,28,65].

2. Resolved motion control: In this type, robot controller commands the manipulator's end effector to move in a desired cartesian direction in a coordinated position and rate control. Resolved motion means that the motions of the various joint motors are combined and resolved into separately controllable end-effector motion along the desired trajectory described w.r.t. WCS. Resolved motion rate control[77,78], resolved motion accelerate control [51], and resolved motion force control[81] are the major approaches in this category.

3. Adaptive control: The above two type of controller sometimes are inadequate in the non-linear compensations of the interaction force between the various joints because they require accurate modelling of the manipulator dynamics and neglect the changes of load in a task cycle.

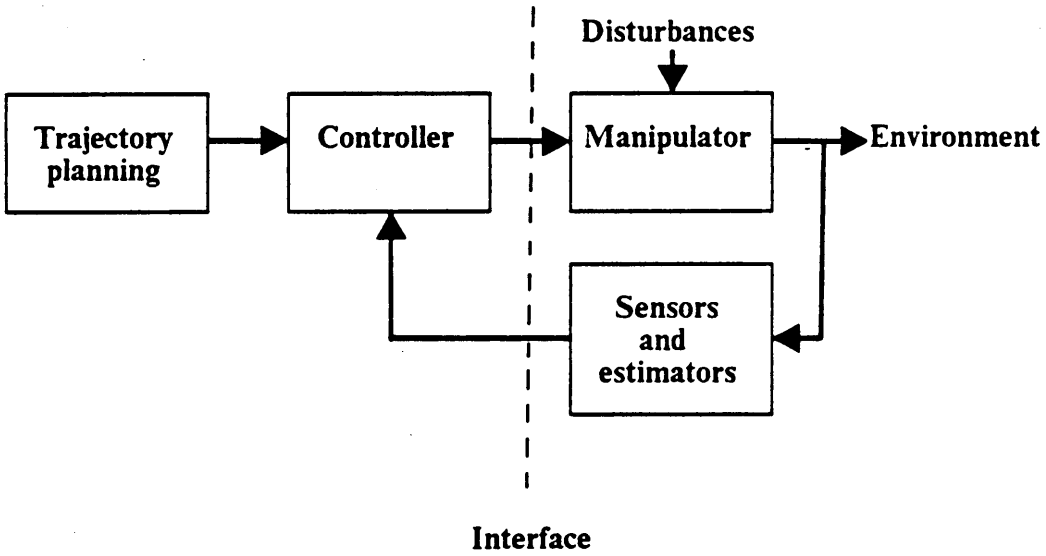


Figure 1. Basic control block diagram for robot manipulator(Adopted from Fu, Gonzalez, and Lee's book[22].

The changes in payload of the controlled system often are significant enough to render the above feedback control strategies ineffective. In order to compensate for the varying dynamic parameters of a manipulator and the different loads that it carries, various adaptive control schemes both in JCS and WCS have been developed. Examples include Model-referenced adaptive control[18], Self-tuning adaptive control[38], Adaptive perturbation control[39,40], Resolved motion adaptive control[41].

1.1.2 Planning the robot motion

Industrial robots are mostly used as spatial positioning devices that are commanded to complete a task by performing a sequence of desired motion segments. Planning such a sequence of motion segments to perform the task is called task planning. Converting an individual motion segment into the time sequence of intermediate configurations is called trajectory planning. In the following sections, the trajectory planning is discussed in section 1.1.2.1 and then the task planning is discussed in section 1.1.2.2.

1.1.2.1 Trajectory planning

Robots are operated by a sequence of operation commands. Among the operation commands, there are two types of operation commands: point-to-point(PTP) type and continuous path(CP) type commands. The PTP type commands specify the starting and ending configurations and the desired joint velocities, whereas the CP type commands specify the starting and ending robot configurations and the intermediate trajectory in between as well as the desired velocity of the end effector. The motion segments generated by the task planning are executed by these two operation commands. It should be noted that the motion segments should be collision-free when executed by the PTP or CP commands. That is, task planning should generate appropriate motion segments

which can be completed by the CP or PTP operations and perform the task well. Therefore, understanding the features of trajectory planning for both type operations is important for task planning. Besides, those features are fundamental to robot simulation.

The trajectory planning converts the desired motion segment into a detailed trajectory defined by the time sequence of intermediate configurations of the robot manipulator between the beginning and ending of the motion. The output of the trajectory planning is a sequence of joint vectors that the end effector will try to follow. The configurations, possibly together with their first and second time derivatives, are then sent in succession to the robot control system as reference inputs. Robots try to achieve the successive intermediate configurations and eventually follow the desired trajectory approximately. This procedure depends heavily on the robot's controller. Due to the different specifications of CP and PTP commands, the trajectory planning may be discussed by one type for PTP operations and the other type for CP operations. They are discussed in Section 1.1.2.1.1 and in Section 1.1.2.1.2 respectively.

1.1.2.1.1 Trajectory planning for PTP operations.

PTP operations are widely used in spot welding, material handling, loading and unloading of machines, and simple assembly tasks. The trajectory planning for PTP operations has the starting point configuration, $\underset{\sim}{q}_s$, ending point configuration, $\underset{\sim}{q}_f$, and specified joint velocities, $\underset{\sim}{v}_m$, as the command input and a couple of constraints listed below are taken into account:

1. the maximum admissible acceleration(a_m), the maximum admissible deceleration(d_m) and maximum admissible velocity(v_{max}) at each joint, and
2. the joint value and work space limitation.

The velocities of the end-effector at the starting and ending points are supposed to be zero. When using the PTP commands, the programmer should note several points listed below.

1. The command input should meet the constraints listed above.

2. The end effector's real trajectory described w.r.t. W.C.S. is unpredictable. Thus, the task should only require that the manipulator's end effector reaches the specified starting and ending configurations and the joint velocities meet the assigned joint velocities.

To minimize the motion time, the controller will move each joint from $\underset{\sim}{q}_s$ to $\underset{\sim}{q}_f$ as fast as possible and stop at the end-point. Each joint of a manipulator moves to its ending position as fast as possible without exceeding a_m, d_m, and v_{max}. The ideal trajectory for a joint can be described by a set of trapezoid motion-time profiles shown in Figure 2 on page 13. which can be described w.r.t. either the JCS or the WCS[43]. As shown in Figure 2, trapezoid motion-time profile consists of three regions: the initial segment[0, t_1] which is based on maximum acceleration, the intermediate segment[t_1, t_2] which is based on maximum desired velocity(v_m), and the final segment [t_2, t_f] which is based on maximum deceleration. The trapezoid motion-time profile can be described by Eq. (1-4)-(1-6).

$$q(t) = \begin{cases} q_s + \dfrac{a_m t^2}{2}, & 0 \le t \le t_1. \\ q(t_1) + v_m(t - t_1), & t_1 \le t \le t_2. \\ q(t_2) + v_m(t - t_2) + \dfrac{d_m}{2}(t - t_2)^2, & t_2 \le t \le t_f. \end{cases} \qquad (1-4)$$

where

$$q(t_1) = q_s + \frac{a_m}{2} t_1^2$$

$$q(t_2) = q(t_1) + v_m(t_2 - t_1)$$

$$\dot{q}(t) = \begin{cases} a_m t, & 0 \le t \le t_1. \\ v_m, & t_1 \le t \le t_2. \\ v_m + d_m t, & t_2 \le t \le t_f. \end{cases} \qquad (1-5)$$

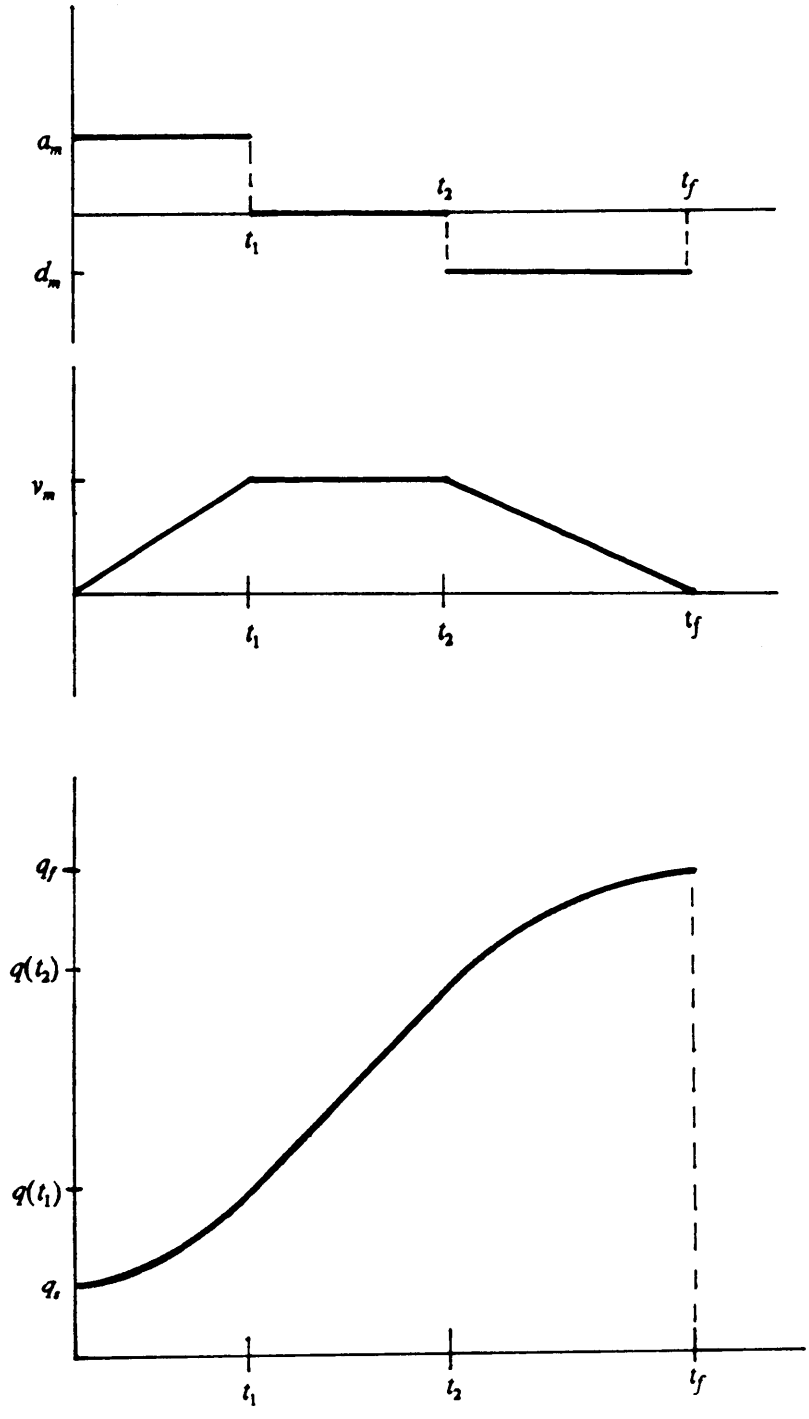


Figure 2. Ideal trajectory of an individual joint under PTP command.

$$\ddot{q}(t) = \begin{cases} a_m, & 0 \le t \le t_1. \\ 0, & t_1 \le t \le t_2. \\ d_m, & t_2 \le t \le t_f. \end{cases} \qquad (1-6)$$

If the joint increment, D, is less than $\frac{v_m}{2}(\frac{1}{a_m} + \frac{1}{d_m})$ then the trajectory of this kind of short move can be described by the triangle motion-time profiles as shown in Figure 3 on page 15. These profiles can be described by the following equations Eq.(1-7)-(1-9).

$$q(t) = \begin{cases} q_s + \frac{a_m t^2}{2}, & 0 \le t \le t_1. \\ q(t_1) + \frac{d_m}{2}(t - t_1)^2, & t_1 \le t \le t_2. \end{cases} \qquad (1-7)$$

$$\dot{q}(t) = \begin{cases} a_m t, & 0 \le t \le t_1. \\ v + d_m t, & t_1 \le t \le t_2. \end{cases} \qquad (1-8)$$

$$\ddot{q}(t) = \begin{cases} a_m, & 0 \le t \le t_1. \\ d_m, & t_1 \le t \le t_2. \end{cases} \qquad (1-9)$$

where

$$v = \frac{(\frac{2D}{(\frac{1}{a_m} + \frac{1}{d_m})})^{\frac{1}{2}}}{} \le v_m$$

$$t_1 = \frac{v}{a_m}$$

$$t_2 = \frac{v}{|d_m|}$$

However, for controllers designed to have smooth motion in order to have small vibration, all joints are scheduled to reach the destination simulataneously. Therefore, only the joint with the longest moving time is moved by the maximum specified velocity as described above. The motion-time profiles of all other joints will be scaled down according to the joint with maximum required motion time.

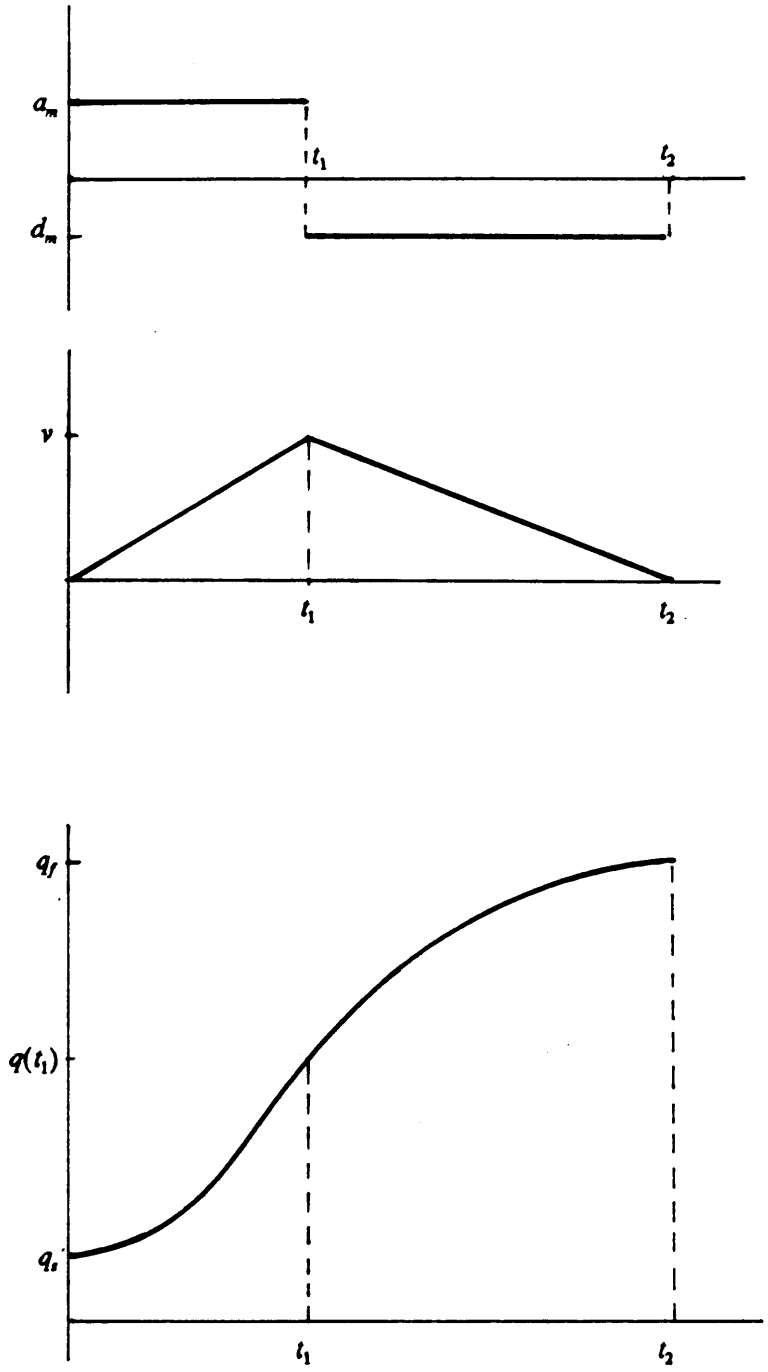


Figure 3. Ideal trajectory of an individual joint under PTP command(short move).

1.1.2.1.2 Trajectory planning for CP operations

The trajectory planning for CP type operation is complicated in command calculation since the end effector must be moved toward the ending-point along a specified trajectory. In CP commands, the robot configurations at starting point, ending point, and intermediate trajectory are usually presented in the end effector's coordinates described w.r.t. the WCS. However, the robots control the motion by joint values. Thus, the world coordinates, $\underline{s}_s$ and $\underline{s}_f$, need to be transformed to joint values, $\underline{q}_s$ and $\underline{q}_f$. Therefore, except in cartesian-type robots, inverse kinematics is needed by the controller.

Industrial robots interpolate the desired path of trajectory into a sequence of intermediate points which divide the trajectory into small sections. Then the robot tries to reach these intermediate points successively and eventually moves the end effector approximately along the specified trajectory. Therefore, the major task of trajectory planning is interpolating the specified trajectory. The interpolation algorithms specify a series of intermediate points between the starting point and ending point along the required trajectory described w.r.t. the WCS. These points should be within the working space of the manipulator and the motion through them should be subjected to spatial velocity and acceleration constraints from the starting-point to the ending-point.

There are still some strong constraints existing for trajectory planning. Since the trajectory interpolating is restricted to the WCS, there are several drawbacks[9].

1. Two nearby robot configurations in the WCS may have corresponding far apart joint vectors. If these two configurations are in successive points along trajectory, the manipulator will move wildly as it passes this discontinuity, or singularity.
2. The trajectory described w.r.t. the WCS usually does not consider physical limitation on joints. Therefore, it might be necessary to unwind a joint by 359 degrees rather than advance it by 1 degree.

3. The inverse kinematics may give several answers. Continuous functions of joint position need to be ensured by taking care the transformation.

The straight line path is frequently specified for a CP command and most industrial robots have the capability to trace a straight line. There are several reasons why a programmer or task programming system might choose straight line paths for end effector to follow(robot motion). First, straight line paths are relatively predictable and easily visualized. Second, shortest paths, though not necessarily the fastest, are given. Third, straight line paths occur in many application cases, such as continuous path welding. Finally, uniform motion along straight line paths minimizes the inertial forces on the end effector and the object it might be carrying.

Whitney[78], Paul[60], and Taylor[71] have introduced their approaches to the straight line trajectory planning techniques. They linearly interpolate the positions of end effector from the starting point to the destination and smoothly change the orientation at the same time. The techniques for straight line trajectory form a basis from which the trajectory for achieving other space curves, such as circle, conics, etc, can be developed. One way is piece-wise linear approximations to space curves[60,71]. The other way, not yet developed, is to generalize the analysis underlying straight line to achieve the corresponding space curve.

It is known that the speed can not be specified as fast for CP commands as for that of PTP commands; otherwise the accuracy to the required trajectory will be reduced. This can be explained as follows. The spacing between intermediate points, the accuracy to the required trajectory, and the velocity at which the robot moves all depend heavily on the computational speed of the interpolation algorithm and inverse kinematics algorithm. Shorter execution time for these algorithms permits a shorter sampling time of the control loop. Shorter sampling time also permits smaller sections when dividing the the trajectory so that the positional accuracy will increase. However, with a specified sampling time, small sections will cause slow robot tracing traveling speed. On the other hand, high traveling speed will cause bigger sections and low accuracy to the required trajec-

tory. Therefore, the task planner should understand the capability of the assigned robot and specify appropriate velocity for CP commands.

1.1.2.2 Task planning

Tasking planning generates the sequence of motion segments to perform the task, and the trajectory planning considers the detailed motion planning for an individual motion segment. The robot task planning can be classified into two groups, on-line manual teaching and off-line task programming. They are discussed in Section 1.1.2.2.1 and Section 1.1.2.2.2, respectively.

1.1.2.2.1 On-line manual teaching

On-line manual teaching is the simplest and most frequently used task planning technique in the industry. Teaching is done by moving the robot manually until the desired robot configuration is reached. The commands of these manual motions are given by the operator, who uses a series of functions offered by a control box or teach pendant. When the desired configuration is reached, the operator stores the configuration and the intermediate trajectory by the available functions, if required, into the computer memory. This process is repeated for each required configuration until the task program is completed. The operator checks of the recorded program by playing the whole sequence and changes the intermediate configuration if necessary. Finally, the velocities for each motion are specified and the task planning is finished.

In manual teaching the operator handles everything in task planning, such as collision avoidance, cycle time, force exerted on the object, etc. Simplicity is the advantage of manual teaching, whereas the main disadvantage is that the equipment is tied up in the programming stage.

1.1.2.2.2 Off-line task programming

The off-line task programming can be classified into two levels: one is manipulator-level programming; the other is the task-level programming. Their difference is whether the programming

specifications are based on the manipulator or the task. The manipulator-level programming may utilize the commercial robot languages as a tool. However, the paths still need to be generated off-line. Task-level programming is still under development. Both levels of off-line task programming are introduced in the following two subsections.

Manipulator-level programming

The commercially available robot programming languages in use today, such as AL, VAL, RPL, TEACH, HELP, AML, etc, provide instructions to move the manipulator, read the sensor, send output signals, and provide many other instructions to simplify task planning. The user instructs the end effector in English-like commands and is not concerned with the motion of the individual joints. These motions are calculated by the language processor, which can be either a complier or an interpreter. Motion instructions in programming languages contain the following types of data:

1. Positional information: starting position (usually the current position at which the command is performed) and ending position of each motion described w.r.t. W.C.S.
2. Operation type: CP or PTP type
3. Velocity
4. Function to be performed at a point, such as a time delay, tool manipulation, and sending or receiving signals
5. Gripper status - closed or open.

Since the motion instruction in those commercial robot languages specifies the motion in end effector's position described w.r.t. the WCS, utilizing them can be treated as the manipulator-level programming. The user needs to plan the sequence of robot path without collision before programming. The paths can be generated by on-line teaching. In this case, the robot language is just a tool to help task programming and actually that is the achievement of the available commercial robot languages by now. The advantage of off-line programming is not really obtained. However, off-line path planning is still under development and actually it is the key part of off-line task programming.

Task-level programming

The existing systems require that the tasks be specified by a sequence of motions, rather than by the world states to be achieved. The goal of task level robot programming systems is to allow task specifications by means of symbolically described states, leaving it to the system to plan the required motion. For this level of programming system, task planning would transform the task-level commands to manipulator-level commands. To do these transformations, the task planner must have a description of the objects being manipulated, the task environment, the robot carrying out the task, the initial state of the environment, and the desired final state. The output of the task planner would be a robot program to achieve the desired final state when executed in the specified initial state. The task planning can be divided into three phases: world modelling, task specification, and manipulator program synthesis.

1. World modelling:

The world model for a task must contain the following information:

1. geometric descriptions of all objects and robots in the task environment;
2. physical description of all objects, e.g. mass and inertia;
3. kinematic descriptions of all linkages of robot manipulators;
4. descriptions of robot characteristics, e.g. joint limits, acceleration bounds, and sensor characteristics.

2. Task specification:

A model state is given by the configurations of all the objects in the environment; tasks are actually defined by sequences of states of the world model. The sequence of model states needed to fully specify a task depends on the capabilities of the task planning program. The ultimate task planning program might need only a description of the initial and final state of the task. This is the goal of much of the artificial intelligence research on automatic problem solving[58]. In the foreseeable future, however, task planning will continue to need significant information about intermediate states, but it can be expected that a much more detailed robot program will be produced.

The proposed method to task specification by a sequence of model states is the specification by a sequence of operations[44,45]. Thus instead of building a model state of an object in its desired configuration, we can describe the operation by which it can be achieved. The description should be object-oriented, not robot-oriented. Most operations also include a goal statement involving spatial relationships between objects. Autopass (Automatic parts assembly system) is an experimental task-level language developed by IBM for assembly tasks, that adopts an operation-oriented approach to task specification.[27,44].

3. Manipulator program synthesis:

The synthesis of a manipulator program from a task specification is the crucial phase of task planning. The major steps involved in this phase are grasp planning, motion planning, and error detection. The output of the synthesis phase is a program composed of grasp commands, a sequence of motion specifications, and error tests. This program should be in a manipulator-level language for a particular manipulator and is suitable for repeated execution without re-planning.

Grasping is a key operation in manipulator programs since it affects all subsequent motions. The grasp planning should take care of:

1. where to grasp objects without collision with other objects,
2. what grasp configurations to keep objects stable, and
3. appropriate force generated during motion and contact with other objects.

Once the object is grasped or no objects are needed to be grasped at some application, generating motions to reliably achieve the desired goal of operation is the next process. The principal goal of these motions is to reach the destination without collision. Therefore, planning a motion is a problem of collision avoidance. More explicitly, a sequence of motion segments is needed to be synthesized to achieve the goal of the operation and every individual motion segment is collision-free, which can be programmed in manipulator-level language as described previously. At this stage,

the same problem, off-line path planning, is encountered as before. It is clear that this is an unavoidable problem in off-line task planning. Details about off-line path planning will be discussed in Chapter 2.

Since the actual state of the world will differ from the world model and the uncertainty can not be avoided totally, the synthesized program should have the capability to detect error and to either correct it or discontinue the action. Error detection and correction in manipulator programs is a very difficult problem. Not much research has been done on this topic[45].

In summary, the task-level programming for robots is still in the experimental stage. No system with the above described functions is already functional on the real robot system. However, this is a good area for research and many researchers are working on it.

1.2 Thesis Outline

Although many robots are already working in plants worldwide, robotics is still an immature area and in need of more research efforts. From the application point of view, off-line task planning can save a lot money for those robot users and make the robot more beneficial and attractive to industry. Off-line path planning is one of the most important problems of off-line task planning. This is already demonstrated in the discussion above. My research direction will focus on the improvement of off-line path planning. In the next chapter, the off-line path planning will be discussed and literature review will be presented.

This thesis consists of four chapters following Chapter 1. In chapter 2, discussion and literature review are done in the broad area of path planning first and then in the specific topic of interest, simulation for path verification. After the searching procedures described in Chapter 1 and Chapter 2, the objective of research, developing a general robot path verification simulation system, GRPVSS, is brought out in Chapter 3. The proposed methodology to achieve the objective is also presented in this chapter. Chapter 4 introduces the structure and algorithms of GRPVSS. Chapter 5 concludes with the summary of methodology and algorithms of GRPVSS and discussions about computation complexity, further extension, and application of GRPVSS.

Chapter 2 LITERATURE REVIEW

In Chapter 1, the general background of robotics is briefly introduced. The research interest is also directed toward the improvement of off-line path planning. In this chapter, the literature on off-line path planning are reviewed first. Two major steps of path planning are path generating and path verification. Currently, the existing proposed off-line path generating algorithms are still in the experimental stage, and the path verification is more important in practical application. Simulation, a widely used technique, is a good tool to test the off-line generated paths. Therefore, the concentration is put on discussing robot simulation systems for path verification and the related literature.

This chapter is organized into two sections. Section 2.1 presents the world modelling and two major steps of path planning. Section 2.2 is focused on the simulation for path verification purpose and the related literature. It is divided into three consecutive topics on:

1. necessary characteristics of simulation system dedicated for path verification purpose,
2. existing simulators, and
3. algorithms of trajectory tracing and those of collision detection.

For consistency, the notations used in the original literature are unified.

2.1 Robot off-line path planning

Off-line path planning has two major steps: off-line path generating and path verification. In path planning, one should take into account several issues as the planning objectives, such as collision avoidance, minimizing cycle time, minimizing actuator's energy cost, etc. Among all these issues, collision avoidance is the most fundamental one because serious damage may occur to equipments and operators whenever collision happens. Modelling the robot's working world is a prerequisite for path planning and significantly affects path planning algorithms. Therefore, world modelling is discussed first in Section 2.1.1, which is followed by the discussion of path generating and path verification in Section 2.1.2 and Section 2.1.3, respectively.

2.1.1 World modelling

The path planning algorithms work on the models of environments in which robots work. The world modelling is fundamental to the path planning algorithms and the geometric description of all objects in the working environment is critical in developing the path planning methods. Furthermore, the characteristics of the world information which includes (1) incomplete vs. complete priori knowledges and (2) static vs. dynamic world significantly affect the feasibilities of path planning methods and may require additional sensory inputs and a more elaborate control system. The geometric models that have been utilized are introduced in Section 2.1.1.1 and the characteristics of world is discussed in Section 2.1.1.2.

2.1.1.1 Geometric primitive models

In order to solve the path planning problems, the space occupied by each object in the robot working world needs to be represented by geometric models. The computation complexity of the path planning algorithm depends heavily on the model. Therefore, the characteristics of the applied models strongly affect the algorithms. Usually, the robots and all objects in the world need to be modelled by a set of primitive models(patterns) before they are further modelled into the assigned models for a certain path planning algorithm. From the application viewpoint, it is impractical to exactly describe the shapes of objects because the mathematical formulation of the exact shapes is too tedious. Besides, it is not necessary to have the exact shapes described because the robot's accuracy is not absolute which will make the exact description meaningless.

Prior approaches to collision-free path planning in three-dimensional space have been to model the objects in the world by a set of polyhedra, cylinders, or spheres. Pieper[62] adapted planes, cylinders, and spheres as the basic model elements. The use of cylinders and spheres doesn't approximate long and sharp objects very well and much of the free space occupied by parts of the cylinders and spheres is wasted for planning purposes. Besides, intersection functions are often nonlinear, which involve square roots or transcendental functions and are time consuming to solve. Udupa[73], Leozano-Perez [47,48], Brook[9], Luh and Campbell[52] utilized the polyhedra as the basic model elements that result in linear intersection functions. However, the disadvantage of wasted free space still exists.

2.1.1.2 Characteristics of world information

The characteristics of world information can be inspected in two ways: (1) incomplete vs. complete priori knowledge of the robot's world, and (2) static vs. dynamic world. They are discussed in the following two paragraphs.

Priori knowledge are the dimensions, positions, and orientations of all objects in the robot's world as functions of time. If the incompleteness is in the form of small uncertainty in poses of objects, the world is said to be *uncertain*[29]. This happens frequently in factory tasks. In such case, trajectories are commonly generated off-line and executed by an intelligent program which uses force control, compliance, and vision to make minor modifications to the trajectories[10]. If the incompleteness is in the form of unknown objects in the world or totally unknown poses of objects, the world is said to be *unstructured* [29]. In such cases, a large portion of the trajectories may have to be generated on-line, after a description of the world has been obtained using complex sensory processing for recognition, description, and localization.

A dynamic world means the robot's working world changes over time, while the static world means the environment remains stationary. Changes in the world may be due to: (1) motions of objects not caused by the robot, or (2) the appearance or disappearance of objects from the world. In the dynamic world, the world information must be updated once it changes.

So far, most of proposed path-planning methods are developed under the static worlds with complete priori knowledge. Besides, only one robot is assumed to work in the environment.

2.1.2 Path generation

With the robot world model, generating an optimal path to move robot from one configuration to another configuration without collision and meeting certain motion criteria is the goal of path generating. Methods are derived to generate a collision-free path from starting robot configuration to ending robot configuration without considering the constraints in dynamics. For most prior developed algorithms, the path generating requires the following major steps.

1. World representation and acquisition

For different algorithms, they need different models to describe the workspace. From the primitive models of the robot and objects in the world, they are transferred into the desired models for certain algorithms. Then, the world representation is obtained. The proposed models to describe the world in three dimensions include:

- Surface-based CAD models[3,63],
- Swept volumes [7,69],
- Cellar array[68],
- Octrees[14,32,55],
- Analytic surface equations[46,47,48,54].

2. Mapping world space to searching space

The search for collision-free paths happens in a searching space in which collision won't happen. The representation might be the same as the world representation. However, they are often different. For instance, the following search space representations are different from world representation: configuration spaces[46,47,46], generalized cone free spaces[8,9], voronoi-based spaces[12], medial axis free space[64], etc. When the search space representation is different from the world representation, a procedure that maps the world to the search space is required.

3. Path searching

In this stage, a set of searching techniques are utilized to find collision-free paths in the searching space which meet the objectives, such as the shortest paths. Utilized searching techniques included A^* algorithm[58], hill climbing[29], etc.

Although the path generation is described in three steps separately, their tasks often overlap and actually they are a sequence of works. The techniques used in one step are designed to incorporate with techniques used in other steps. So far, all these proposed algorithms are still in the exper-

imental stage. Although they may work in some special cases, current algorithms that find optimal paths among three dimensional space with obstacles are typically not efficient.

2.1.3 Path verification

For the off-line generated path, it is necessary to verify its feasibility and quality. There are two ways to verify the planned path. One is on-line verification by running the planned paths on the real system; the other is off-line path verification using the computer. It is dangerous to directly apply the off-line generated path on a real system without off-line path verification because serious damages may happen if there is any unexpected collision. Therefore, the off-line path verification is recommended before the planned paths are applied on the real systems. Robot simulation is a good technique for off-line path verification. The robot simulator simulates the robot behavior as it moves its end effector through the assigned paths. When the concentration is put on verifying the off-line generated paths to be collision-free, the collision detection during the robot motion is the major concern of the robot simulation. In Section 2.2, the robot simulation for collision-free path verification will be discussed in detail.

2.2 Robot simulation for collision-free path verification

In this section, the robot simulation for collision-free path verification is discussed. First, the necessary characteristics of a simulation system for collision-free path verification are discussed in Section 2.2.1. These necessary characteristics are used as principles in discussing and evaluating the available simulators. The available simulators for path verification purpose are reviewed individ-

ually in Section 2.2.2. Finally, a summary about the algorithms for trajectory tracing and collision detection is presented in Section 2.2.3.

2.2.1 Necessary Characteristics of simulation for collision-free path verification

A good simulation system for collision-free verification purpose has to have the following three characteristics.

1. The simulation system can reflect the real manipulator trajectory to an acceptable extent.
2. The simulation system efficiently detects collisions once they occur.
3. The simulation system requires feasible computation time.

It should be noted that they influence one another significantly, although they are stated individually. For example, more accurate robot trajectory needs more computation time. These three characteristics are discussed individually in the two topics that follow: trajectory tracing and 100% collision detection guarantee.

2.2.1.1 Trajectory tracing

Undoubtedly, the robot motion simulation needs to reflect the real manipulator trajectory to an acceptable extent. Only with precise trajectory is the collision detection meaningful. In simulation systems, there are two types of approaches to trace the manipulator motion or generate the intermediate configurations during the motion. One is based only on kinematics and the other is based on both kinematics and dynamics. Usually, the simulator utilizing the former one is called kinematic simulator and the simulator utilizing the latter one is called dynamic simulator. They

are conceptually described in the following two paragraphs and will be further discussed on how they do the trajectory tracing in Section 2.2.3.

A. Kinematic simulation system

In a kinematic simulation system, simulation of robot motion is based on the manipulator kinematics described in Chapter 1. The motion of the robot manipulator considers only the kinematic relationship between the robot links. The simplicity is the major advantage of this type simulator. However, since the simulators do not consider the force effects, the deviation caused by force constraints or effects are not reflected in the simulated manipulator motion.

B. Dynamic simulation system

Manipulator dynamics describes the relationship between the joint forces/torques and joint positions, velocities, and accelerations. Beside the kinematic relationship between manipulator links, the robot trajectory is generated by applying the direct dynamics given the generated joint forces/torques from the employed control law. Dynamic simulation considers the dynamic characteristics of the manipulator-drive assembly and therefore it reflects the manipulator motion more accurately. However, the computation complexity in dynamic analysis increases the simulation complexity enormously in comparison with the kinematic simulation.

2.2.1.2 100% collision detection guarantee

Under the assumption that the simulated robot motion trajectories reflect the real trajectories, the collision detection algorithm is performed at each intermediate step. In order to guarantee 100% collision detection, the collision detection algorithm should possess the ability to detect any collision happened during the intermission between the two successive simulation instants. Since the collision detection is based on the models which represent robot manipulator and all entities in the

robot working world, the utilized models strongly affect the efficiency of collision detection algorithm. The proposed collision detection algorithms will be reviewed later in this chapter.

2.2.2 Robot simulators

A lot of researches have been focused on the robot simulation. They can be categorized into two types according to their purposes. One aims at supporting the design of a robot itself and mainly treats problems of kinematics and dynamics on manipulator and control system. The other aims at supporting the motion planning for off-line teaching or programming and mainly treats problems of motion analysis and synthesis, as well as its graphical technique, collision detection and collision avoidance problems. The robot simulation for collision free path verification belongs to the second category. Several simulation systems for collision-free path verification are discussed below.

Bonney et al.[5] developed a GRASP- Graphical Robot Application Simulation Package to support the generation and verification of robot operations on an interactive graphical terminal. The geometric modelling describes objects by 3-D models. Each model is defined by the size and shape of its primitive models and a detailed specification of their interconnections. The primitive models include cuboid and prism. Robot's models are contained in the data files which also include the information of the robot's kinematic parameters. GRASP is a kinematic simulation system; dynamics is not considered. The homogeneous transformation method is used to describe the structure and kinematics of manipulator linkage system. Available options of operation types include PTP, straight line, and paths given in terms of mathematical functions. Their paper doesn't clearly show how the trajectory tracing is done.

The collision detection method in GRASP checks geometric interference for each intermediate robot configuration. The geometric interference detection is performed between all primitive models of the robot and those of all objects in the working environment. Due to the characteristics of

polyhedra, the primitive models intersect if and only if at least one surface of one polyhedra intersects one surface of the other. (GRASP assumes that the robot would not be completely contained by the obstacles.) Since GRASP only does the static checking in interference detection, the user needs to carry out a rough visual check first during a display of the simulation process. In a three dimensional graphic display, it is hard to detect collision visually. Thus, there is no guarantee that collision can be detected between the simulation steps. Besides, only kinematics is considered in motion analysis. Dynamic factors are ignored in the simulated motion. However, graphical simulation also gives users a clear demonstration of the robot motion.

Kitajima[36] developed a robot simulation system based on a model of general-purpose structure developed in the CAD system for a general assembly unit with motion. A hierarchical network structure is employed to model the robot. Also, the shape parameters, kinematic parameters, and transformation matrices between links are also generated from the model. It is a kinematic simulator. Several primitive models, such as cuboid, cylinder, prism, pyramid, etc, are proposed. The collision detection problem is solved by checking if any interference exists between every pair of primitive models of the robot and objects in the workspace. The interference detection between two primitive models is performed by checking the existence of any intersection points between every edge of one primitive model and every surface of the other. This simulator has the following advantages:

1. It can simulate all types of robot with open-loop or close-loop type links.
2. It offers the inverse kinematic transformation equations from the the hierarchical structure model and uses the numerical method to solve them. The computation complexity increases as the link number increases.

Since the primitive model includes cylinders, the interference equation may not be linear, and the computation complexity is higher. This simulation system also has the deficiency of kinematic simulation because it doesn't consider dynamic effects. Besides, the interference detections are

performed only at each simulation step. There is no guarantee that collision happening within the simulation interval can be detected.

Sungurtekin and Voelcker[69] presented approaches to do graphical simulation and automatic verification of NC machine program. They utilized the swept volume of the machine body between two simulation steps to detect the collision. Under this condition, the collision happening during simulation intermission won't be missed. However, if this approach is utilized at a multi-linked robot in 3-D space, the computation complexity of computing swept-volume would be enormous.

2.2.3 Algorithms for trajectory tracing and collision detection

Several proposed simulators with path verification capability have been reviewed in the previous section. They are discussed individually and the concentration of the last section is to individually show the characteristics of the simulator. This section focuses on the proposed approaches that achieve the required characteristics of a simulation system. The algorithms used in the proposed simulators to trace the robot trajectory and detect the collision are summarized in Section 2.2.3.1 and Section 2.2.3.2.

2.2.3.1 Algorithms for trajectory tracing

The validity of a robot simulation system depends on the precision level of reflecting the real robot system behavior. Good robot simulators closely imitate the trajectory of a real system. The manipulator trajectory tracing can be based on kinematics, or both kinematics and dynamics. The proposed algorithms for the two type simulators, i.e. kinematic simulators and dynamic simulators, are reviewed in section 2.2.3.1.1 and Section 2.2.3.1.2 respectively.

2.2.3.1.1 Kinematic simulators

Various kinematic simulators have been developed[5,43,57,59]. In kinematic simulators, generating intermediate configurations of robot motion is based on the method of coordinate transformation. The homogeneous transformation method with DH convention described in Chapter 1 is a common method used to describe the structure and kinematics of the manipulator with open-loop links[17,43,59]. For manipulators with close-loop links, additional modification would be needed. Various forms of such modifications has been described by Sheth and Uicker[67], Derby[17], Kitajima[36]. Here, the concentration is on the robot manipulator with open-loop links. Although some minor differences occur in the notation, most approaches adapt the notation, as the convention used in Chapter 1, modified from the notation of Paul[61]. This coordinate transformation forms the basis of the simulation. The Eq. (1-1) (1-3) in Chapter 1 are applied to do the coordinate transformation.

$$T_n^0 = A_1^0 \cdots\cdots A_{n-1}^{n-2} A_n^{n-1} \qquad n \geq 2 \qquad (2-1)$$

$$P_0 = T_i^0 P_i \qquad (2-2)$$

Whenever there is movement, the T_i^0 transformation matrix is obtained by applying the joint vector q of the new position on Eq.(2-1). Using Eq.(2-2), the coordinates of every point of the manipulator links w.r.t. local coordinate frame can be transformed to the coordinates described w.r.t. WCS. Then, the whole manipulator configuration described w.r.t. WCS at a certain simulation step is built up.

The emulation of robot motion is achieved by sequentially simulating a sufficient large number of intermediate robot configurations. Due to the different types of motion operations(CP and PTP), the trajectory generating mechanism for these two types of operations is different. In most litera-

ture, the emphasis has been placed on the features of simulation programs developed, not on the method to generate the trajectory. The reported methods are described below.

For PTP operations, the motion can be specified by the initial and final joint vector, $\underline{q}_i$ and $\underline{q}_f$. In most simulators[43], the intermediate configurations are formed by linearly interpolating the joint-value increment w.r.t. the path length. Therefore, the j-th intermediate robot configuration of a total n step simulation has joint values given by

$$\underline{q}(j) = (\underline{q}_f - \underline{q}_i)(j/n) \qquad (2-3)$$

Simulators of this type don't simulate the real trajectory well because they assume the joint velocities are constant throughout the trajectories. The assumption definitely doesn't reflect the true situation because infinite acceleration is caused under this assumption.

For CP operations, the motion can be specified by initial and final positions and orientations of the end effector($\underline{p}_i$, $\underline{o}_i$ and $\underline{p}_f$, $\underline{o}_f$) and trajectory described w.r.t. WCS. Since the robot configurations are constructed by the joint values, the inverse kinematic solution of the simulated robot should be solvable. Most industrial robots have the capability to perform the straight-line CP type operations. In a simulator constructed by Leu et al[43], the intermediate end effector's positions are generated by linearly interpolating the straight-line path under the assumption that the robot's end effector will exactly follow the path. The orientation is assumed to be smoothly changed to the final orientation. The inverse kinematic transformation is utilized to generate the joint vector from the position and orientation of the end effector and the robot configuration at a certain simulation step is obtained.

For further function applications in some kinematic simulators or dynamic analysis, the velocity analysis is necessary. The following widely-known formulations are the relationship between velocity($\underline{\dot{s}}$) and the derivative of joint vector($\underline{\dot{q}}$).

$$\underline{s} = \begin{bmatrix} \underline{P} \\ \underline{o} \end{bmatrix} \quad , \quad \dot{\underline{s}} = \begin{bmatrix} \dot{\underline{P}} \\ \underline{w} \end{bmatrix} \tag{2-4}$$

$$\dot{\underline{s}} = J_q \ \dot{\underline{q}} \tag{2-5}$$

$$\dot{\underline{q}} = [J_q]^{-1} \ \dot{\underline{s}} \tag{2-6}$$

where

$\dot{\underline{p}}$ is the velocity of the end effector described w.r.t. WCS.

$\underline{w}$ is the rotation rate vector of the coordinate frame at end effector along the three axes of the WCS.

J_q is a $6 \times n$ matrix called the Jacobian matrix whose elements are dependent in the joint vector $\underline{q}$.

Nemec and Lenarcic[57] utilized the following formula in their simulator to derive the velocity of the end effector of an n-linked manipulator. It is known that the relationship of robot configuration described w.r.t. the JCS and the WCS may be defined by a set of six independent trigonometrical equations.

$$\underline{s} = P_q(\underline{q}) \tag{2-7}$$

The Jacobian matrix can be derived by the following formula.

$$J_q = \left[\frac{\partial P_q(\underline{q})}{\partial q_1}, \frac{\partial P_q(\underline{q})}{\partial q_2}, \ldots, \frac{\partial P_q(\underline{q})}{\partial q_n} \right] \tag{2-8}$$

This method is good for Catersian robots with only three links. However, the derivation of Eq.(2-7) is very complicated for a manipulator with six links.

Koren presented a simple method in his book[37] to derive the $\dot{\underline{s}}$ based on the contribution of $\dot{\underline{q}}_j$ to the vector $\dot{\underline{s}}$. $\dot{\underline{P}}_j$ and $\underline{w}_j$ are the contributions of $\dot{\underline{q}}_j$ to $\dot{\underline{P}}$ and $\underline{w}$, as shown in Figure 4 on the following page.

For a revolute joint:

$$\dot{\underline{P}}_j = (\underline{u}_j \times \underline{b}_{jh})\dot{q}_j \qquad (2-9-1)$$

$$\underline{w}_j = \underline{u}_j \dot{q}_j \qquad (2-9-2)$$

For a prismatic joint:

$$\dot{\underline{P}}_j = \underline{u}_j \dot{q}_j \qquad (2-10-1)$$

$$\underline{w}_j = 0 \qquad (2-10-2)$$

where

$\underline{u}_j$ is pointing to the rotation axis of $\dot{q}_j$ which coincides with the Z_{j-1} axis of the local coordinate frame at link j-1.

$\underline{b}_{jh}$ describes the distance between joint j and position of the end effector.

Subsequently, the column of J_q are obtained according to their definition(Eq. (2-5)).

$$J_{qJ} = \begin{bmatrix} \underline{u}_j \times \underline{b}_{jh} \\ \underline{u}_j \end{bmatrix}, \quad \textit{for a revolute joint.} \qquad (2-11)$$

$$J_{qJ} = \begin{bmatrix} \underline{u}_j \\ 0 \end{bmatrix}, \quad \textit{for a prismatic joint.} \qquad (2-12)$$

$$J_q = \left[J_{q,1}, J_{q,2}, \ldots, J_{q,n}\right] \qquad (2-13)$$

The two vectors $\underline{u}_j$ and $\underline{b}_{jh}$ can be calculated from the homogeneous transformation matrices that are derived for direct kinematic solutions. This method is quite appropriate for kinematic simula-

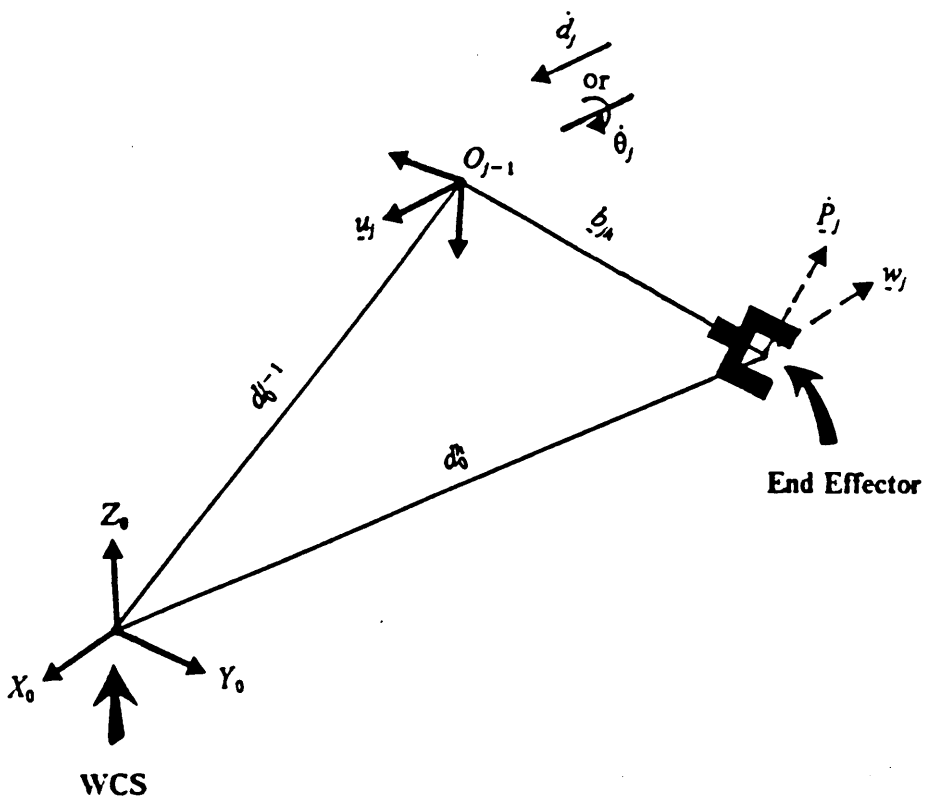


Figure 4. The contribution of the joint velocity of component j to the velocity vector.

tors because all required inputs can be easily derived from the outputs of the homogenous transformation. However, this method only generates the Jacobian at the origins of the coordinate frames.

Paul and Rosa[59] proposed another method to solve the velocity of a point $\underline{P}$ fixed at link m of an n-linked manipulator. The derived formulas are listed below.

$$\dot{\underline{P}} = D_m \underline{P} = [\dot{x}_1, \dot{x}_2, \dot{x}_3, 1]^t \qquad (2-14)$$

where

$$D_m = \sum_{k=1}^{m} Q_k^* \dot{q}_k = D_{m-1} + Q_m^* \dot{q}_m , \qquad 2 \le m \le n. \qquad (2-15)$$

$$D_1 = Q_1^* \dot{q}_1 = Q_1 \dot{q}_1 \qquad (2-16)$$

$$Q_k^* = T_{k-1}^0 Q_k [T_{k-1}^0]^{-1} , \qquad 1 \le k \le n. \qquad (2-17)$$

$$Q_k = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} , \quad \textit{for a revolute joint.} \qquad (2-18)$$

$$Q_k = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} , \quad \textit{for a prismatic joint.} \qquad (2-19)$$

T_{k-1}^0 : Transformation matrix from local coordinate frame at link k-1 to the WCS.

The positioning velocity Jacobian for point ($\underline{P}$) at link m of an n-linked manipulator is

$$\dot{\underline{P}} = J_m \dot{\underline{q}} \qquad (2-20)$$

$$J_m = S\left[\underline{Q}_1^* \underline{P},\ \underline{Q}_2^* \underline{P},\ \ldots,\ \underline{Q}_m^* \underline{P},\ \underset{3 \times (n-m)}{0,\ 0,\ \ldots,\ 0}\right] \qquad (2-21)$$

where

$$S = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad (2-22)$$

2.2.3.1.2 Dynamic simulator

The block diagram shown in Figure 5 on the following page is adopted from Walker and Orin[76]. It well describes the structure of a typical dynamic simulator. The desired trajectory is furnished by the motion design section. With the desired destination($\underline{q}_d$), the controller generates torques($\underline{\tau}$) on the manipulator's links by actuators according to the control law. Then direct dynamics is applied to compute the joint acceleration($\ddot{\underline{q}}$) of the manipulator by the exerted torques($\underline{\tau}$). Through motion integration, the actual trajectory is determined. The dynamic simulation utilizes the dynamic analysis to simulate the real trajectory of a manipulator given certain control laws. Walker and Orin[76] presented four methods for solving the inverse dynamic equations. They also compared them w.r.t. their computational efficiencies. The computation complexity of solving dynamic equations is enormous even with the most efficient methods. The execution time of the four different methods to simulate a five second trajectory of a six-linked manipulator using CDC Cyber-175 is shown in Table 1 on page 43. The results in Table 1 show how expensive dynamic simulation can be.

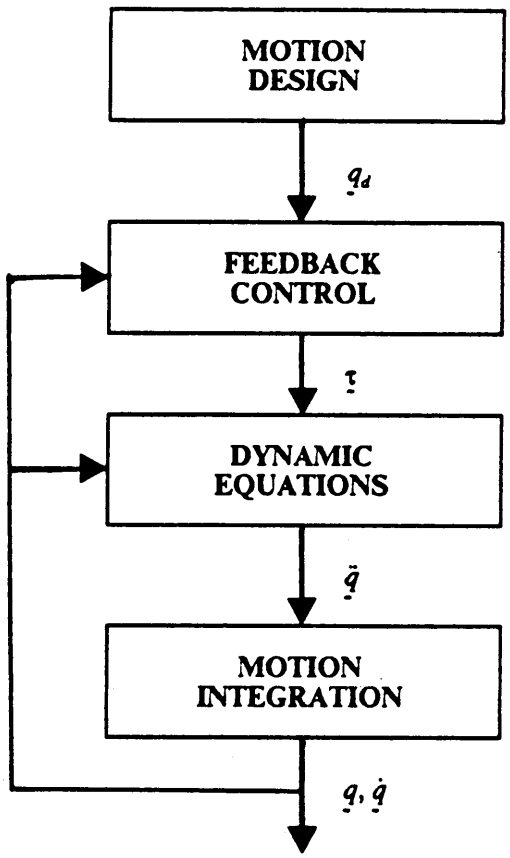


Figure 5. Dynamic simulator block diagram(adopted from Walker and Orin's paper[76]).

Table 1. Execution time of four different methods of a five second trajectory(adapted from Walker and Orin's paper[76])

Method	Execution Time
1	65.663 sec.
2	50.712 sec.
3	37.395 sec.
4	61.910 sec.

Leu and Mahajan[43] presented a graphic simulator based on robot kinematics and dynamics. The kinematic part was already stated in the previous section. The problem studied in their dynamic motion simulators is stated as follows. The simulator finds the motion of each joint, i.e. displacement vs. time, given a manipulator with its dynamic model and the associated mass properties, the initial and goal robot configurations, the constrained joint velocities and accelerations, and the constrained joint forces/torques. The resulting motion minimizes the travel time and satisfies the imposed constraints.

In Leu and Mahajan's work, the robot motion is assumed to be a joint-interpolated motion so that the movements of all the joints can start and end at the same time and the required joint forces/torques are bounded. In order for a joint motion to be completed in the shortest possible time without violating the acceleration constraints, it is necessary that the motion starts with the constrained acceleration and ends with the constrained deceleration. Thus, the displacement, velocity, and acceleration functions for one joint are assumed to be ideal trapezoid functions as shown in Figure 2 on page 13 and Figure 3 on page 15. Required motion time is the longest travel time of all joints, and the maximum velocity and acceleration of other joints should be scaled down accordingly. The forces/torques required for the motion data $\underline{q}(t)$, $\underline{\dot{q}}(t)$, and $\underline{\ddot{q}}(t)$ process are solved by inverse dynamics and are compared with the constrained forces/torques. If the constrained values are exceeded, the maximum velocities and accelerations are scaled down again. The scaling process involves an iterative procedure comparing the required and the constrained joint forces/torques recursively. Of course, the long execution time for inverse dynamic analysis cannot be avoided in this simulator. This approach doesn't include the controller's characteristics in the simulation. It simply applies the ideal trapezoid profiles, and the advantage of the dynamic analysis is lost. However, the simplicity is its advantage.

2.2.3.2 Collision detection algorithms

To detect collisions is to decide if any robot configuration in its motion course involves sharing a given region of space with other objects at the same time. The problem can be viewed from two aspects, interference detection and collision detection. Interference detection checks the intersection between stationary geometric models, whereas collision detection considers a situation where at least one object is in motion. Since the intersection checking is performed between two geometric representations, collision detection may be viewed as a sequence of interference detections with appropriately defined static models and time intervals. Thus, the basic problem of collision detection in the robot motion simulation appears to be interference detection at each simulation step. In the following two subsections, the proposed algorithms for geometric interference detection and collision detection are reviewed in the stated order.

2.2.3.2.1 Interference detection

In the past, most of the related work which requires interference detection have used polyhedra as the primitive model of objects[6,15,46,54,73]. These polyhedra can be described by the coordinates of and adjacent relationships among its vertices. This approximation reduces the computation complexity of solving the intersection points by its linear intersection equations.

Comba[15] utilized convex objects with differentiable surfaces to model the objects in his approaches. The expression that represents the bounding surfaces of all the objects are combined into a single psedo-characteristic function $G(x,y,z)$. The $G(x,y,z)$ function has the property that the region with $G \leq 0$ is a close approximation to the region where the objects intersect.

Maruyama[54] modelled the objects as polyhedra or unions of convex polyhedra. He employed the smallest spheres($\hat{S}_i$) which contain the corresponding objects and the maximal spheres($\check{S}_i$) which are contained in the objects to do the preliminary interference test. If the distance between the centers of $\hat{S}_i$ and $\hat{S}_j$ of two objects is longer than the sum of their radii, they are strongly separated.

If the distance between the centers of $\check{S}$ of two objects is smaller than the sum of their radii, they strongly intersect. Maruyama then utilized minimal boxes, the smallest rectangular parallelpipes which contain the objects, to do the interference detection between the pairs of objects which are not strongly separated. Although he utilized the polyhedra to model the objects, he still employs one object as a box to do the interference detection. This feature increases the wasted free space and the chance of false conclusions. The interferences may be detected when they do not occur.

Boyse[6] also utilized polyhedra to model objects. However, he distinguished the objects to three types: solids, surfaces, and containers. Surface representation is provided because in many applications only the surface geometry is of interest. Containers are different from solids only in the voidity of their interior. Intersections are detected between polyhedra by checking if an edge of one polyhedron of an object insects a surface of one polyhedron of another object. Along with the linear intersection equations, this feature makes his approach efficient and widely used in application. Since he also employs the container as the primitive model, he also has to detect whether the interference joint is inside the boundary.

Ahuja et al.[1] proposed a method in which the non-interference is detected by checking the non-overlapping projections of two polyhedra over a collection of planes. Two polyhedra are non-overlapping if the projections on at least one plane appear to be non-overlapping. However, for any given number and choice of planes, non-interfering objects may be mistaken as overlapped because projections overlap. Increasing the number of planes may only decrease, but not eliminate, the probability of error detection. Moreover, the projection procedure of one ploy hedron takes at least $O(N \log N)$ time, where N is number of vertices of the polyhedra. Besides, non-convex polygons in projection planes still have to be decomposed into convex polygons in order to utilize Shamos's algorithm[66] to find the intersection of two convex polygons. This method is quite conservative and suffers information loss from the reduction in dimension. Besides, the computation complexity is higher.

Ahuja et al[1,2] also utilized an octree representation of objects to facilitate the interference detection. They regard the entire space as one block in the beginning. If a block is completely contained within the object, the representation of the object is sought. Otherwise, it is divided into eight octants each of which is treated similarly. This splitting procedure continues until all the blocks are either completely within the object, called black block, or completely outside the object, called white block, or a block of minimum size (limit of resolution) is reached, called gray block. This recursive subdivision feature allows a tree description of the occupancy of the space. Each mode may be one of the three types of blocks. Thus, the interference detection may be executed on the tree description of the pair objects. The objects intersect if there exists in the two trees at least one pair of corresponding nodes of which one is black and the other is black or gray. Thus, the existence of interference can be determined by traversing the two trees in parallel and comparing the corresponding nodes. The execution time is proportional to the actual number of nodes traversed. For a moving object, the octree representation needs to be renewed once it moves. This feature decreases this algorithm's feasibility in robot simulation because robots are usually in motion.

2.2.3.2.2 Collision detection

In this section, the approaches to test for collision between a moving object and a stationary object are reviewed.

Boyse[6] detected the collision between two polyhedra by detecting a collision of every edge of the moving polyhedra with every face of the other stationary one. The collision detection algorithm considers the following two possible situations.

1. Moving edge contacts face interior: Since contact must occur at an end-point of the edge, collision can be detected by determining the locus of each vertex of the moving polyhedron

and by checking whether either one of these space curves intersects the surfaces of the stationary polyhedron.

2. Moving edge contacts face boundary: Collision is detected by checking the boundary of the face to see if it intersects the surface swept by the moving edge.

The computation of generating the locus of vertices and surface swept by the moving edge is quite complicated for the rotation or translation along an axis which is moving in 3-D space. For robot manipulators, this condition is a common case. However, this approach can perfectly detect the collision once it happens.

Ahuja[1] extended the interference detection algorithm to accommodate for the collision detection between a moving object and a stationary object. Since interference detections are applied to a snapshot of the continuously varying spatial configurations of the moving object, care must be taken to ensure that no collision will occur between any two successive executions of the interference detection. Thus, when each interference detection is applied, it is essential to find out whether or not some objects are too close and might collide before the instant when the next interference detection is applied. Therefore, the moving objects are expanded by a thickness D. The safe distance D between any two objects is proportional to the speeds and relative locations of the objects involved and $D \geq V_i T_i$, where V_i, T_i are the speed of the moving object and time interval to apply interference detection at instant i. Eventually, the collision detection is performed by executing periodic interference detection on modified objects. This approach simplifies the collision detection. However, this method may detect non-existing collisions due to the growth of objects, and therefore deciding the time interval (T_i) becomes important. The above maximum allowable displacement of the the moving object does not consider the shapes and distribution of obstacles. Further, how to decide the safe distance is an important issue which was not discussed by Ahuja et al..

In this chapter, off-line path planning has been introduced. Simulators with path verification capability have been reviewed individually. A summary of the proposed algorithms of trajectory

tracing and collision detection has been presented. From all these reviews, the objective of research and the methodology to achieve the objective will be addressed in the next chapter.

Chapter 3 RESEARCH OBJECTIVE and METHODOLOGY

In the previous two chapters, the general background of robotics has been briefly introduced and robot simulation for path verification has been discussed in detail. Through those discussions, developing a general graphic robot simulation system with path verification capability emerges to be the objective of this research. The research objective is described in Section 3.1 and the methodology to achieve the objective is presented in Section 3.2.

3.1 Objective Statement

From the application point of view, task planning is one of the most important works on the plant floor. Currently, the task planning is performed by on-line manual teaching, which is expensive because the machines are tied up at the planning stage. Off-line task planning is the expectation of robot users and manufacturers, and path verification is one unavoidable step in path planning.

Robot simulation offers a good tool to verify the off-line generated path. So far, no efficient algorithm that fully verifies the path has been developed and applied in the reported simulators.

The objective of this research is to construct a General Robot Path Verification Simulation System(GRPVSS) which has the following characteristics:

- generalized for all industrial robots with open-loop linkage
- reflecting the real robot trajectory to acceptable extent
- 100% collision detection guarantee
- simplification for feasible computation time
- graphic animation.

The working environment of the simulation system is restricted to one robot and static objects(obstacles). The result can be extended to multi-robots and moving objects. The extension will be discussed in Chapter 5.

3.2 Methodology

The objective has been identified in the previous section and the methods used to achieve the objective are described in this section. The hierarchical structure of GRPVSS is shown in Figure 6 on the following page. As shown in Figure 6, the simulation procedures are grouped into five steps: world modelling, path input, trajectory generation mechanism, collision detection, and graphic an-

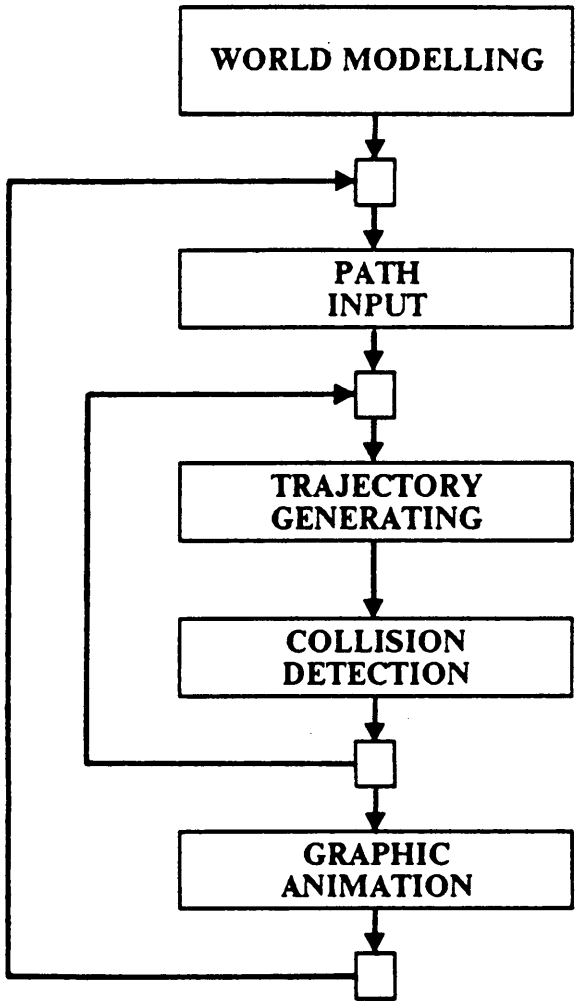


Figure 6. The hierarchical structure of GRPVSS

imation. In the world modelling step, all information about the simulated robot and the objects in the world is input in terms of the representing models. Then, the user specifies a path which is composed of a sequence of motion segments. Each motion segment is completed by a motion command which can be either a CP command or a PTP command. For each motion segment, the robot's intermediate configurations are sequentially generated by the trajectory generation mechanism of its operation type and collision detection is applied at each intermediate configuration. The path is performed by sequentially completing all the motion segments of the specified path. When all motion segments of the path are completed, the graphics animation is displayed to show the user clear images of the motion. The approaches used in each step are discussed in the following five subsections respectively and major algorithms will be discussed in Chapter 4.

3.2.1 World modelling

World modelling, the first step in the simulation procedure, models everything in the robot's working space. World modelling has been discussed in Chapter 1 for task planning. Here, the necessary information of this simulation system includes:

1. Kinematic parameters of all links
2. Shape parameters of all links of manipulator and those of the objects in the environment
3. Robot system characteristics, such as manipulator joint limits, joint values of the manipulator at home position, joint acceleration bounds, and the optional inputs(motion-time profile of every joint and the solution of its inverse kinematics)
4. User specified tolerance of inaccuracy.

The skeleton of a robot is constructed by homogeneous transformation with the DH convention as mentioned in Section 1.1.1.1.1. This procedure needs the kinematic parameters(θ, d, α, and a) of every joint and transformation matrices are obtained by Eq.(1-1)-(1-3) in Chapter 1. For each

manipulator, there are constraints on joint value variables, i.e. the θ values for revolute joints and *d* values for prismatic joints have upper limits and lower limits. These values are important to determine the workspace and to judge whether a certain manipulator configuration is achievable. Every robot has a home position and its configuration is a necessary input. Usually, the motion starts from home position.

In order to describe the space occupied by the manipulator and objects in the working space, shape parameters should be defined. For the simplicity of modelling and efficient interference detection, the primitive models are selected to be convex polyhedra. All links of the robot manipulator and all objects in the working environment are approximated by convex polyhedra or unions of convex polyhedra that contain the manipulator link or object. A typical convex polyhedron used as primitive model is shown in Figure 7 on the following page. Each wire-frame convex polyhedron model(see Figure 7) has 16 vertices. Vertex 1-8 form a ploygon in one plane, while vertex 9-16 form a polygon with the same shape in a plane(surface B) parallel to the last one. Thus, the shape parameters of the manipulator links are the coordinates of vertices described w.r.t. the link coordinate system(LCS), which should be recorded following the order of their numbers. The order of their numbers defines the relation of the vertices. With the transformation matrices T_i^0, the whole shape of a robot can be described w.r.t. the WCS. The shape parameters of the objects in the workspace are the coordinates of vertices described w.r.t the WCS. This modelling approach wastes some free space. However, simplicity is obtained. Besides, power wires and communication wires usually exist around the manipulator's links. Those stuff can be modelled by the convex polyhedra.

Several characteristics of the simulated robot are necessary inputs for the GRPVSS. The actuators which drive the movement at joints have maximum torques. These maximum torques can be converted to the maximum accelerations and decelerations of corresponding joints. Moreover, the motion-time profiles, i.e. position-vs-time functions and velocity-vs-time functions, of manipulator joints can be derived for certain robot manipulator. If those profiles are not available, GRPVSS will utilize ideal trapezoid profiles as shown in Figure 2 on page 13 and Figure 3 on page 15. These

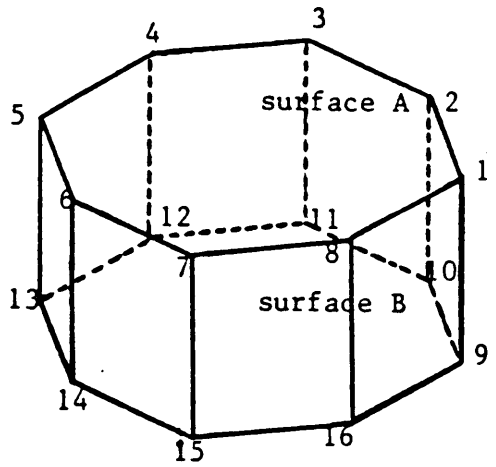


Figure 7. The primitive model: a typical convex polyhedron.

functions will be utilized to generate the trajectory of PTP type operations. If the paths are of CP operations, the inverse kinematics solutions should be available. Therefore, the user should supply subroutines to offer the inverse kinematics solutions of the simulated robot given the joint vector of the manipulator.

No robot can move exactly on the specified trajectory. Its accuracy depends on the mechanical inaccuracies, controller's performance, and robot system resolution. GRPVSS needs an user specified tolerance(Δ) described w.r.t. WCS for robot motion. There are two reasons for using this value; one reason is to compensate for the inaccuracy inherent to the robot; the other is to serve as the safety factor for collision detection. For the second reason, every polyhedron which forms manipulator links is expanded by a thickness of the amount of tolerance. The reason why the links need to be expanded will be illustrated in Section 3.2.4. The robot expansion algorithm will be discussed in the next chapter.

3.2.2 Path input

The off-line generated path needs to be separated into a sequence of motion segments. The GRPVSS can simulate both PTP type and CP type straight-line operations. Therefore, for each motion segment, the user needs to specify operation types. The necessary inputs for both operation types are listed below.

The inputs of PTP type command include:

1. the joint vector($\underset{\sim}{q}$) or coordinate of end effector($\underset{\sim}{s}$) described w.r.t. to the WCS of manipulator configuration at destination(The starting point is the robot's configuration while the command is performed.)
2. the required joint velocities.

The inputs of CP type command include the manipulator configuration at destination described w.r.t. the WCS.

3.2.3 Trajectory generation mechanism

The robot's motion trajectory between the current position and the destination is necessary for motion simulation and important for collision detection. For simulation, trajectory generation means generating a sequence of intermediate configurations of the motion. As discussed in the previous chapter, based on the method used for trajectory generation, the simulation system can be divided into kinematic simulators and dynamic simulators. The dynamic simulators provide motion paths that are close to their corresponding real ones. However, dynamic simulations are very time consuming and not practically applicable. Therefore, GRPVSS doesn't include the dynamic analysis and it is constructed to be a kinematic simulator. In order to make the trajectory closer to the real one, the motion time profiles, that are interpolated to give the motion path of PTP type operations, are employed. Due to the two types of motion operations, the trajectory generation methods are classified into two types, one type for PTP operations and the other for CP operations. They are discussed in the next two subsections individually.

3.2.3.1 Trajectory generation for PTP operations

The trajectory of PTP type operation is determined by the motion-time profiles of the simulated robot. Each joint has a set of corresponding motion-time profiles which are joint position vs. time functions and joint velocities vs. time functions. With these motion time profiles and the increment of joints($\underline{q}_f - \underline{q}_s$) of a certain PTP operation, the joint positions at a corresponding time are known. That is, the joint positions of intermediate trajectory are determined. When the motion time pro-

files of the simulated robot are not available, GRPVSS utilizes the ideal trapezoid profiles, which have been discussed in Section 1.1.2.1.1 as the substitute.

A block diagram in Figure 8 on page 59 shows the processes of the intermediate configuration generation for PTP opeations. From the input of PTP type commands, the increments of joints ($\underline{q}_f - \underline{q}_s$) are known. Either the motion-time profiles of the simulated manipulator, or the ideal trapezoid functions are used to determine intermediate trajectory. Therefore, the joint vector($\underline{q}$) is determined by a time factor and the increment of the motion simulation is real time. For each step of simulation, the motion-time profiles are applied to obtain the joint values ($\underline{q}$). Now, the problem of how to determine the resolution of simulation, i.e. the time increment per simulation step, should be considered. Since the purpose of this simulation system is to detect the collision whenever it occurs and the computation time is desired to be as short as possible, the resolution of simulation is preferred to be as large as possible so long as the collision happened in the intermission can be detected. The resolution determination is under the principle that the movement δl (described w.r.t. the WCS) of every origin of LCS located at joint i should not be bigger than $\mathrm{MAX}\{\mathrm{MIN}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{\min}\}$, where θ is the smallest angle of the obstacle hulls, $D_{\min}$ is the distance between the nearest obstacle and the corresponding moving link and L is the smallest thickness of all obstacles (the derivation of this principle will be discussed in Section 3.2.4.). Thus, the resolution for joint i can be determined by $\delta t_i \le \frac{\delta l_i}{v_i}$, where v_i is the speed of joint i. The resolution determination chooses the shortest needed time increment of all joints as the time increment to the next step. Then the next intermediate configuration's joint vector is obtained by putting the real time at the next step into the motion-time profiles. The homogeneous transformation method is applied to obtain coordinates of the vertices of the convex hulls described w.r.t. the WCS. As a result, the space occupied by the manipulator is obtained and is ready for collision detection.

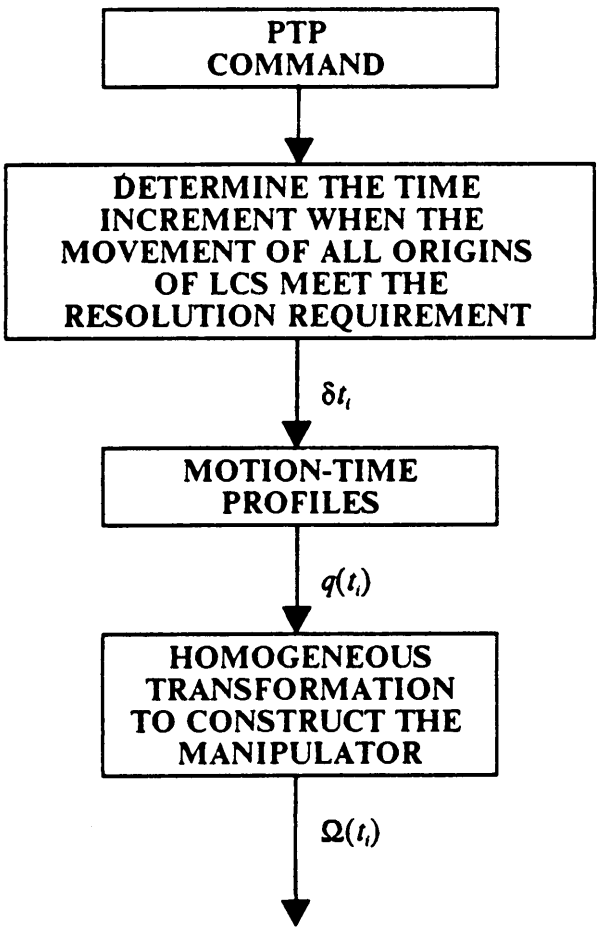


Figure 8. The structure of intermediate configuration generating for PTP operations.

3.2.3.2 Trajectory generation for CP operations

For CP type operations, the intermediate trajectories are assumed to exactly follow the specified trajectories. That is, the end effector of the manipulator exactly traces the specified path with desired orientation (if specified). Under this condition, the intermediate motion configurations are fully determined. Since the purpose of this simulation is to detect collision, the velocity is not of major concern. Therefore, the velocity of the end effector is not considered in CP operations. A sequence of intermediate points which are generated by interpolating the path determines the intermediate configurations. However, since the robot configurations are constructed by joint vector, $\underline{q}$, the inverse kinematics solutions are needed. It is known that there is no systematic analytical method to solve the inverse kinematics problems in close form except for the special design described by Pieper[62]. Therefore, GRPVSS does not have the subroutines to solve the inverse kinematics problems of general kinematic structures. The inverse kinematics for the IBM 7545 robot has been furnished. Users have to support subroutines to solve the $\underline{q}_i$ given $\underline{s}_i$ for other robots.

Given the specified path, $C(l)$ where l is the length from the starting point, this path is interpolated into a sequence of intermediate points. While the end effector is at one intermediate point, the robot configuration is one simulation step. With the intermediate end effector's position and orientation, $\underline{s}_i$, the inverse kinematics is applied to compute the joint vector, $\underline{q}_i$. Then, the homogeneous transformation constructs the robot configuration described w.r.t. the WCS and it is ready for collision detection.

As described previously, the movement of each joint at each simulation step should be less than $\mathrm{MAX}\{\mathrm{MIN}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{\min}\}$, in order to meet the resolution requirement. The intermediate configuration generation determines the next intermediate position of the end effector with maximum increment without violating the resolution requirement. Thus, the resolution of simulation is determined by the position increment of the end effector instead of the time increment

which is employed by PTP type operations. Detailed resolution determination algorithms will be discussed in the next chapter.

3.2.4 Collision detection

The collision detection algorithms have been reviewed in Chapter 2. The methods proposed by Ahuja et al.[1], Maruyam[54], and Boyse[6] for collision detection and interference detection are modified and extended for GRPVSS. The method expands the moving manipulator's links by a thickness of the tolerance amount to ensure that potential collisions can be detected by simply applying the interference detection between the expanded manipulator's links and obstacles at each simulation step. Here, determining the tolerance and the simulation resolution is very important because of the existence of robot tracking errors. A new approach is developed to determine the maximum simulation resolution which avoids the interference undetected during the simulation intermissions. Therefore, this collision detection method is discussed in two aspects: simulation resolution determination and interference detection.

A. Resolution of motion simulation

In order to have 100% path verification, collision should not occur between two consecutive simulation steps. To assure collision detection, the robot's arms are expanded by a tolerance thickness. (The algorithm of expanding manipulator links will be discussed in the next chapter.) Then, the intrusion amount of an obstacle without collision being detected between two consecutive simulations should be less than the tolerance. The space occupied by the robot links can be considered as a set of spatial-related points. The points that are most critical to have collisions are on the boundary (vertices, edges and surfaces). Therefore, the maximum allowable displacement of 100% verification of a moving link can be determined by considering those points on the boundary. Since the manipulator is expanded by the amount of tolerance value, any point on the manipulator boundary can be treated as a sphere of radius Δ. Although it is more precisely to be considered as

a cubic of the length 2Δ, sphere is used for simplifying the governing rule of the allowable movement of the point. Besides, the sphere are contained by the cubic. Thus, the result is more conservative. As shown in Figure 9 in the following page, the displacement (δX described w.r.t. the WCS) requirement has to be:

$$\delta X \leq \frac{2\Delta}{\cos(\frac{\theta}{2})} \qquad (3-1)$$

where

θ is the smallest angle of the obstacle hulls. $0 < \theta < 180$ for convex hulls.

Δ is the user specified tolerance value.

If δX is bigger than $2\Delta + L$, where L is smallest thickness of all obstacles, the obstacle with the thickness L still could intrude. Therefore, the movement requirement is modified to be

$$\delta X \leq \mathbf{MIN}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L] \qquad (3-2)$$

In the Eq.(3-2), the maximum allowable displacement δX does not consider the distribution of the obstacles in the working environment. That means the obstacles may be at the most critical position. When the distances between the moving link and all obstacles are bigger than the required δX, the obstacles are distributed at place outside the critical distance δX . Thus the the maximum allowable displacement of the moving link is the distance between the moving link and its nearest obstacle. Therefore, the following rule is developed to determine the maximum allowable displacement (δl) to the next simulation step of a moving link in a certain environment at a certain instant.

$$\delta l = \mathbf{MAX}\{\mathbf{MIN}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{min}\} \qquad (3-3)$$

where

D_{min} is the distance between the moving link and the nearest obstacle.

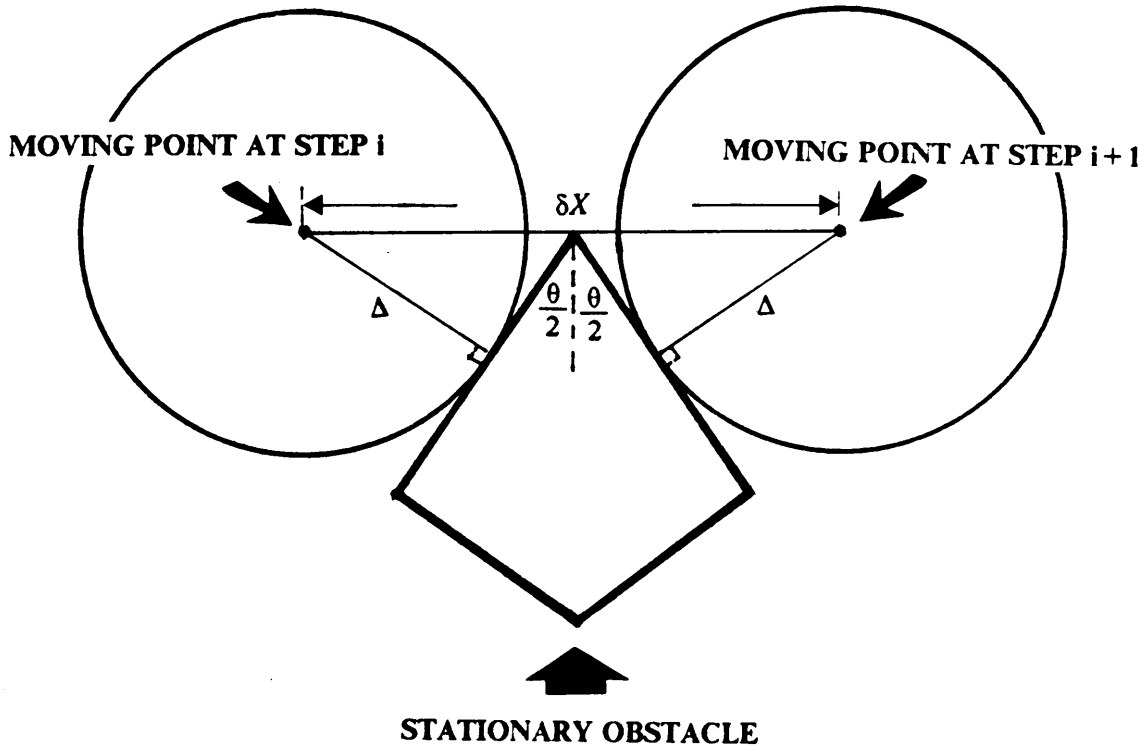


Figure 9. The relationship of intrusions of an obstacle and moving object.

It is observed that different parts of the link may have quite different directions and speeds. In a continuum open-looped linkage mechanism, the ends of the links have the biggest movement. For DH convention, the origins of LCS are located at the links' tips. Therefore, this allowable maximum movement is applied at the origins of LCS. Due to this arrangement, this method guarantees that collision happened on the link tips can be 100% detected.

Tolerance is an important factor of resolution. From Eq.(3-2), it is known that the bigger tolerance is specified, the bigger simulation resolution can be used, and the more conservative result is obtained. That is, there is more chance to have collision detected even when there is no collision. Therefore, choosing an appropriate tolerance is a key issue of resolution. If the result shows collision-free, it is guaranteed that there is no collision in this path. If collision detected, it may not happen in reality.

B. Interference detection

For each intermediate configuration, geometric interference detections need to be performed between expanded links and obstacles(objects in the working space). Geometric interference detection between two solids can be very complicated if these two solids have highly irregular shapes. In order to simplify the computation, the robot's arms and obstacles are approximated by convex polyhedra as defined in the previous section. The whole manipulator arms and obstacles should be contained by those convex polyhedra. Therefore, if intersections occur between the manipulator arm and obstacles, the interference would also occur between convex hulls of the manipulator and convex hulls of obstacles. This approach is conservative because some free space is contained by convex polyhedra However, complexity is reduced and significant computation time is saved.

Because the robot and obstacles are formed by convex polyhedra, interference detection becomes much simpler. If there is interference, at least one edge of one convex hull will interfere with the surface of the other one. Therefore, the only detection needed is the interference between edges and surfaces. Only exception of this rule is when one convex hull is completely contained in another

one. This condition doesn't need to be considered in this case, because the manipulator is impossible to be totally contained in the obstacles in the working space, at least in the beginning. Processes are separated into two parts.

1. Interference detection between edges of convex hulls of manipulator's links and surfaces of convex hulls of obstacles.
2. Interference detection between edges of convex hulls of obstacles and surfaces of convex hulls of manipulator's links.

The second part is redundant. However, for the conservatism, those checkings are also performed. In order to save unnecessary detection between two highly-separated convex polyhedra, prejudgement using distance between centroids is processed. The distance (DIS_{ij}) between the centroids($\underline{P}_{ci}$ and $\underline{P}_{cj}$) of two hulls(i and j) is:

$$DIS_{ij} = \sqrt{(x_{ci} - x_{cj})^2 + (y_{ci} - y_{cj})^2 + (z_{ci} - z_{cj})} \qquad (3-4)$$

where (x_{ci}, y_{ci}, z_{ci}) are the coordinates of centroid of hull i.

If

$$DIS_{ij} \geq D_i + D_j + \Delta \qquad (3-5)$$

where

D_i is the longest distance between the vertices and centroid in convex hull i.

then convex hull i and convex hull j should be collision free within simulation resolution. Detailed interference detection algorithm will be discussed in the next chapter.

3.2.5 Graphic Animation

Animating the robot motion in the screen gives the user a clear image of the simulated path. Through the visual inspection, the user may check the quality of the simulated path. The robot motion animation is achieved by displaying a sequence of intermediate manipulator's configuration. Large number of intermediate configurations gives good image resolution. However, the computation time increases. Since image resolution is not the major concern, the resolution of graphics display is set to be same as the resolution decided for the collision detection. The objects in the GRPVSS are constructed by convex polyhedra defined by the vertices and they are displayed by the wireframe of the convex polyhedra.

The 3-D graphics display program is generated by using the subroutines supported by graPHIGS, a graphics software package published by IBM. At Virginia Tech., this package is installed in VTVM3 of the IBM mainframe computer system. The workstation can be IBM 3250, 5080, Graphical Data Display Manager(GDDM), and Graphics Data Format(GDF) graphics workstations. The supervisor of GRPVSS, which controls the whole processes of the simulation system, interacts with the user to process the functions. In this feature, GRPVSS can be treated as a CAD system.

The methodology of the GRPVSS has been presented. In the next chapter, detailed algorithms used in GRPVSS will be described. The user's guide of GRPVSS is presented in Appendix A.

Chapter 4 The STRUCTURE and ALGORITHMS of GRPVSS

In Chapter 3, the methodology of GRPVSS has been described. In this chapter, the structure of GRPVSS is described in Section 4.1. Then, the major algorithms: algorithms for robot expansion, intermediate configuration generation (including resolution determination) for CP and PTP type operations, and interference detection, are presented in Section 4.2.

4.1 The structure of GRPVSS

The flowchart of GRPVSS is shown in Figure 10 and Figure 11 in the following two pages. Through the simulation system, supervisor controls the whole process of the simulation system. From step 1 to step 3, the user and supervisor communicate to input the necessary information. Most of the information for world modelling is stored in data files that include the robot's kinematic parameters and shape parameters and obstacles' shape parameters, or in compiled execution sub-

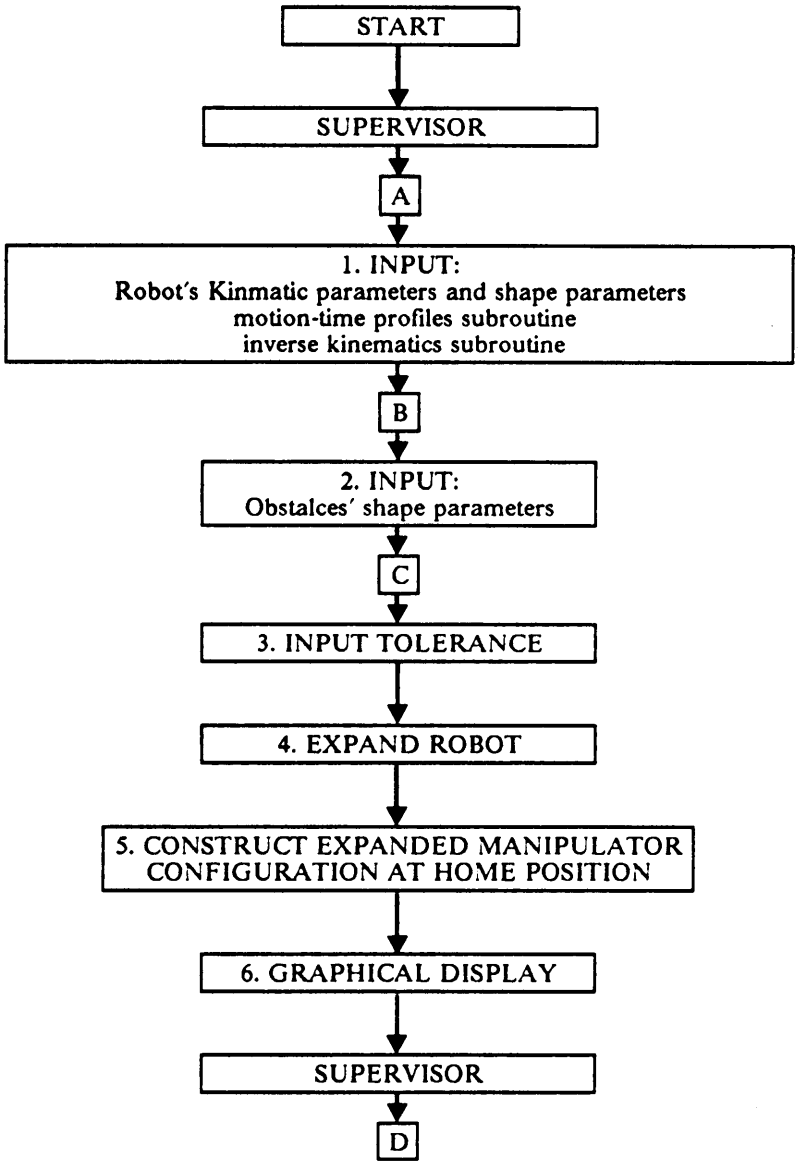


Figure 10. Main flowchart of GRPVSS (Part A).

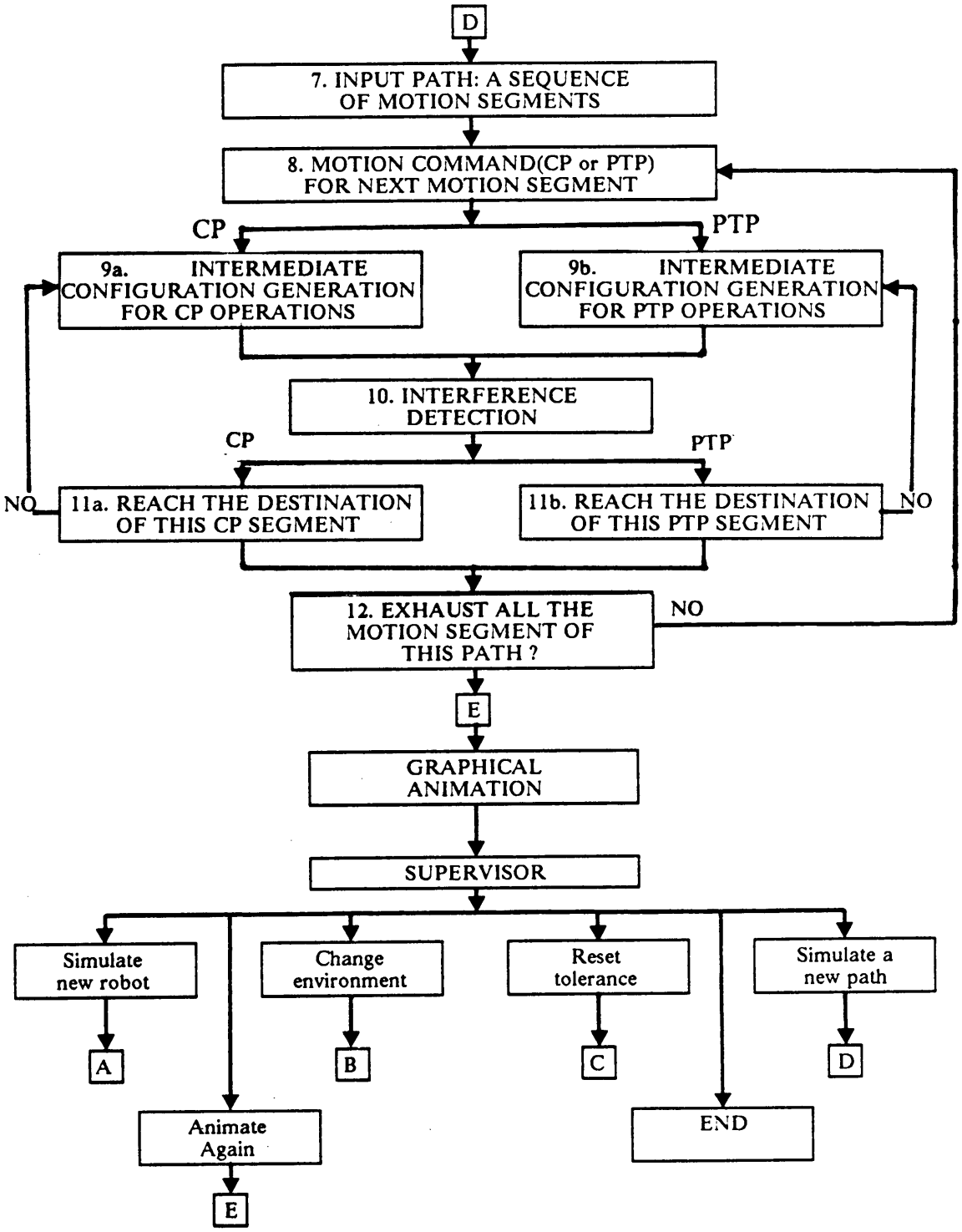


Figure 11. Main flowchart of GRPVSS (Part B).

routine files, including the motion-time profile subroutine and inverse kinematics subroutine. The user assigns tolerance value in step 3.

At the fourth step, the robot expansion algorithm is utilized to expand the manipulator's links a thickness of the amount of user specified tolerance. Since the manipulator's links are modelled by one or several convex polyhedra, the method to expand the robot is to expand every polyhedra of every link the amount of tolerance. The vertices of the expanded convex hulls are the new shape parameters. This robot expansion algorithm is described in Section 4.2.

Based on the kinematic parameters and shape parameters, the homogeneous transformation method is utilized to construct the manipulator's configuration at the home position in step 5 and the configuration is displayed at the terminal in step 6. Supervisor then asks for the described path which is composed of a sequence of motion segments. Each individual motion segment should be specified by either a PTP command or a CP command with its associated specifications. The PTP command's specification includes ending point($\underline{q}_f$ or $\underline{s}_f$ if inverse kinematics subroutine is available). The CP straight line command's specifications is the ending point of end effector($\underline{s}_f$). If the desired path is a space curve, users need to separate it into a sequence of straight-line motion segments. All specifications of the motion segment may be sequentially stored in a data file, or may be interactively specified by the user with supervisor.

With the specification of all motion segments, one by one motion segments are traced sequentially. Step 9 and step 10 generate one intermediate configuration and do the interference detection for that configuration, respectively. For each motion segment, step 9 and step 10 are iterated until the destination is reached.

In step 9, the intermediate configuration generation algorithms are distinguished into two types for CP and PTP operations. Both of these types will be discussed in Section 4.2 in detail. The interference detection algorithm in step 10 will also be discussed in Section 4.2. Steps 8-11 are iterated

until all the motion segments of the specified path are finished. Then, graphical animation will display the whole path on the screen, and any interference point if happened will be shown during the motion.

The processes mentioned above complete the simulation of the desired path. Afterwards, the supervisor takes over to give the user the following options:

1. Simulate new robot: go back to node A.
2. Change environment(obstacles' position): go back to node B.
3. Reset tolerance value: go back to node C.
4. Simulate a new path: go back to node D.
5. Animate again: go back to node E.
6. End the simulation: terminate the simulation.

The major algorithms described previously are discussed in the next section.

4.2 Algorithms of GRPVSS

In this section, the major algorithms: robot expansion algorithm, intermediate configuration generation algorithms for PTP and CP operations, and interference detection algorithm, are discussed in detail.

4.2.1 Robot expansion algorithm

The robot manipulator is approximated by convex polyhedra. Therefore, expanding the robot is equivalent to expanding the convex polyhedra which model the manipulator links. The algorithm to expand a convex polyhedron can be divided into two procedures as shown in Figure 12 on page 73. The first procedure is to generate one parallel line for each edge at surface A and surface B of Figure 7 on page 55. The parallel line is away from the edge by the amount of $\sqrt{2}\,\Delta$, where Δ is the tolerance value specified by the user(see Figure 9 on page 63 and Figure 13 on page 74). The second procedure computes the intersection points of those parallel lines. Finally, the intersection points are the vertices of the expanded hull. The algorithms of these two procedures are described as follows:

1. Generating the parallel line of each edge:

An example as shown in Figure 13 is to compute the parallel line of edge $\overline{P_2P_3}$. Two steps are needed in this procedure.

a.) compute P_0

$$P_0 = P_3 \pm \frac{(P_3 - P_2) \times N}{\|(P_3 - P_2) \times N\|}\,\Delta \qquad (4-1)$$

where

$N = (A,B,C)^t$ is the normal vector of plane Ax + By + Cz + D = 0

The sign(either + sign or − sign) is selected if the requirement, $(P_0 - P_3)\cdot(P_4 - P_3) \le 0$. or $(P_0 - P_3)\cdot(P_1 - P_3) \le 0$. is met.

b.) compute the point P_l

$$P_l = P_0 \pm \frac{N}{\|N\|}\Delta \qquad (4-2)$$

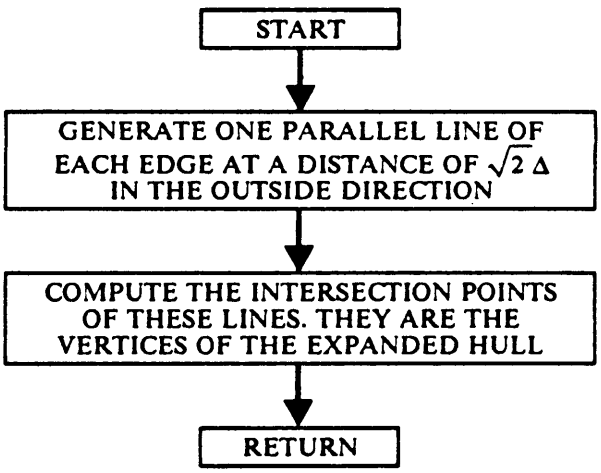


Figure 12. The structure of robot expansion algorithm.

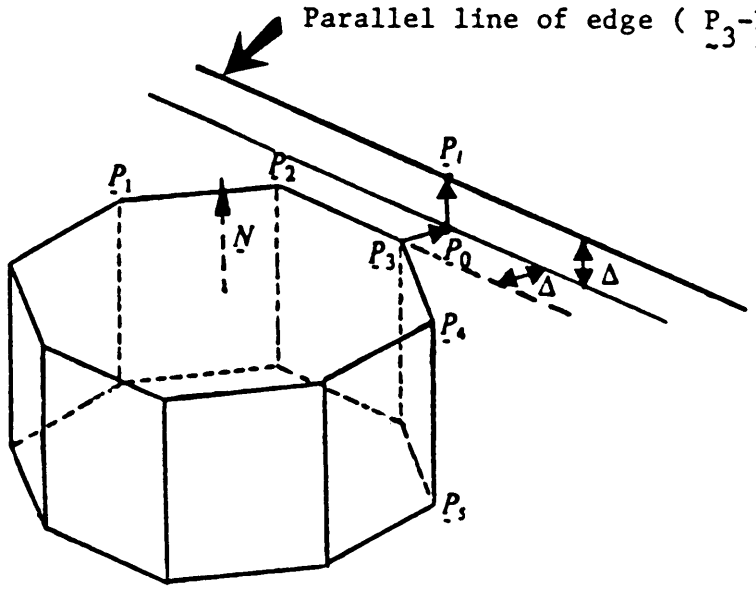


Figure 13. The parallel line of one edge of a convex polyhedron.

+ sign of Eq.(4-2) is selected if 1.) $\underline{P}_5$ and origin are at the different side of the plane, and $\underline{N}$ points to the origin, or 2) $\underline{P}_5$ and origin are at the same side of the plane, and $\underline{N}$ points away from origin.

− is selected otherwise.

The resultant parallel line is: $\underline{P}_1 + \lambda\,(\underline{P}_3 - \underline{P}_2)$, where $\lambda \in R$

2. Robust intersection points computation.

Theoretically, two unparalleled lines on the same surface should intersect, but this may not be true when these two lines are encoded into a finite precision machine. The inherent machine inaccuracy may perturb these two lines into the presentation of two distinct planes. Therefore, instead of solving the intersection point, the method employed calculate the shortest distance between the two lines in space is used. As shown in Figure 14 on page 76, the mid-point of the line segment of the shortest distance, which is perpendicular to both lines, is assumed to be the intersection point. The formula to determine the intersection point of

Line 1: $\underline{P}_1 + \lambda_1\underline{d}_1$, $\quad \lambda_1 \in R$

Line 2: $\underline{P}_2 + \lambda_2\underline{d}_2$, $\quad \lambda_2 \in R$

where: $\underline{P}_1$, $\underline{P}_2$: position vectors

$\underline{d}_1$, $\underline{d}_2$: direction vectors

λ_1, λ_2 : parameters are derived as the following.

The shortest distance between lines 1 & 2 is obtained by minimizing

$$D^2(\lambda_1, \lambda_2) = [(\underline{P}_1 + \lambda_1\underline{d}_1) - (\underline{P}_2 + \lambda_2\underline{d}_2)]^t \;[(\underline{P}_1 + \lambda_1\underline{d}_1) - (\underline{P}_2 + \lambda_2\underline{d}_2)] \qquad (4-3)$$

Partially differentiate Eq.(4-3) by λ_1 and λ_2 .

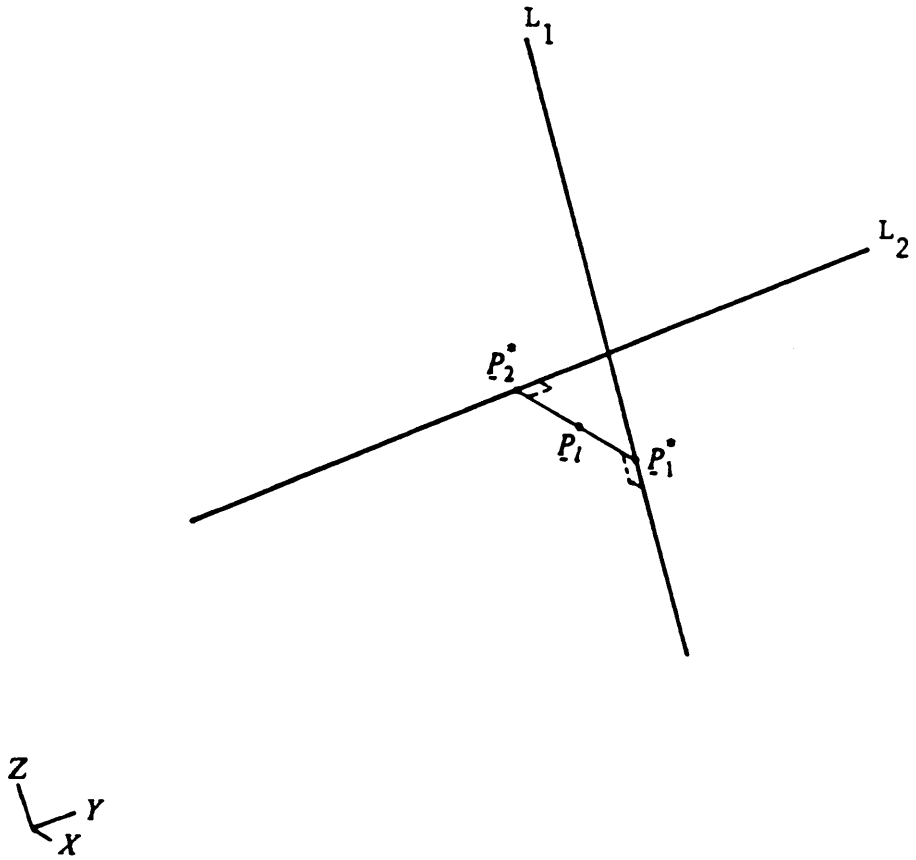


Figure 14. The mid-point of shortest distance.

$$\frac{\partial D^2(\lambda_1, \lambda_2)}{\partial \lambda_1} = 2\underline{d}_1^t \underline{P}_1 + 2\lambda_1 \underline{d}_1^t \underline{d}_1 - 2\underline{d}_1^t (\underline{P}_2 + \lambda_2 \underline{d}_2) = 0 \qquad (4-4)$$

$$\frac{\partial D^2(\lambda_1, \lambda_2)}{\partial \lambda_2} = 2\underline{d}_2^t \underline{P}_2 + 2\lambda_2 \underline{d}_2^t \underline{d}_2 - 2\underline{d}_2^t (\underline{P}_1 + \lambda_1 \underline{d}_1) = 0 \qquad (4-5)$$

Solving the following equations for optimal answers: λ_1^*, λ_2^* .

$$\underline{d}_1^t \underline{d}_1 \lambda_1^* - \underline{d}_1^t \underline{d}_1 \lambda_2^* + \underline{d}_1^t (\underline{P}_1 - \underline{P}_2) = 0 \qquad (4-6)$$

$$\underline{d}_1^t \underline{d}_2 \lambda_1^* - \underline{d}_2^t \underline{d}_2 \lambda_2^* + \underline{d}_2^t (\underline{P}_2 - \underline{P}_1) = 0 \qquad (4-7)$$

Then the answers are:

$$\lambda_1^* = DENO^{-1}[(\underline{d}_2^t \underline{d}_2)(\underline{d}_1^t (\underline{P}_2 - \underline{P}_1)) + (\underline{d}_1^t \underline{d}_2)(\underline{d}_2^t (\underline{P}_1 - \underline{P}_2))] \qquad (4-8)$$

$$\lambda_2^* = DENO^{-1}[(\underline{d}_1^t \underline{d}_2)(\underline{d}_1^t (\underline{P}_2 - \underline{P}_1)) + (\underline{d}_1^t \underline{d}_1)(\underline{d}_2^t (\underline{P}_1 - \underline{P}_2))] \qquad (4-9)$$

where

$$DENO = (\underline{d}_1^t \underline{d}_1)(\underline{d}_2^t \underline{d}_2) - (\underline{d}_1^t \underline{d}_2)^2$$

When DENO = 0, these two lines are parallel. Then, arbitrarily pick λ_1^* and from Eq.(4-5),

$$\lambda_2^* = \underline{d}_2^t \underline{d}_2^{-1} [\lambda_1^* (\underline{d}_1^t \underline{d}_2) + \underline{d}_2^t (\underline{P}_1 - \underline{P}_2)]$$

Therefore, the points which have the shortest distance in line 1 & 2 are:

$$\underline{P}_1^* = \underline{P}_1 + \lambda_1^* \underline{d}_1 \qquad (4-10)$$

$$\underline{P}_2^* = \underline{P}_2 + \lambda_2^* \underline{d}_2 \qquad (4-11)$$

The intersection point, the midpoint of the shortest distance, is:

$$\underline{P}_l = \frac{1}{2}(\underline{P}_1^* + \underline{P}_2^*) \qquad (4-12)$$

From the output of applying this algorithm to do the expansion of IBM 7545 robot on IBM 4341 computer, the shortest distance is less than 3.0E-06 in single precision. This result shows that this method is quite robust and avoids the unnecessary trouble of numerical problems in computers.

4.2.2 Intermediate configuration generation algorithms

The mechanism of trajectory generation of the simulation system generates the intermediate configurations of the simulated motion. The methodology used to generate the trajectory was discussed in Chapter 3. After a specific robot trajectory is determined, the simulation resolution(the increment for each intermediate configuration) is considered. Simulation resolution plays a significant role in the proposed collision detection method. The determination of simulation resolution was also discussed in Chapter 3. Here, the intermediate configuration generation algorithms developed to satisfy the requirements of the given simulation resolutions are discussed. Due to the different methods used to determine trajectory of PTP type operations and CP type operations, the algorithms are separated into two types for two kinds of operations. They are discussed in the following two subsections individually.

4.2.2.1 Intermediate configuration generation algorithm for PTP operations

As discussed in Chapter 3, the increment in motion simulation of the PTP operations is the advancing time domain. Through a resolution determining subroutine for PTP operations, the time increment is obtained. Then, the motion time profiles or ideal trapezoid profiles are applied to obtain the joint vector of the manipulator configuration at the next step. The motion-time profiles and ideal trapezoid profiles which are discussed before, are subroutines of GRPVSS. For a given

time that elapses from the beginning, the subroutine gives the joint position and the joint velocity at that moment. Here, the concentration is the resolution determination for PTP operations.

Resolution determination for PTP type operations

Before discussion, it should be noted that the origin of LCS at link i is located at joint i + 1 for the DH convention. As described before, the movement δl_i of each origin of LCS at link i should be less than $\mathbf{MAX}\{\mathbf{MIN}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{\min}\}$, where θ is the smallest angle of the obstacle hulls, $D_{\min,i}$ is the distance between the nearest obstacle and link i, and L is the smallest thickness of all obstacles. The required time increment for the origin of LCS at link i has been derived as

$$\delta t_i \le \frac{\delta l_i}{v_i} \qquad (4-13)$$

where

v_i is the speed of the origin of LCS i at the current position described w.r.t. the WCS.

Therefore,

$$\delta t_i \le \frac{\mathbf{MAX}\{\mathbf{MIN}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{\min}\}}{v_i} \qquad (4-14)$$

$$\dot{\underline{s}} = \begin{bmatrix} \dot{\underline{p}} \\ \underline{w} \end{bmatrix} = [J]_i \dot{\underline{q}} \qquad (4-15)$$

$$v_i = \|\dot{\underline{p}}\| \qquad (4-16)$$

where

δt_i, $i = 1, 2, \ldots, n$. is time increment needed to meet 100% verification at joint i + 1, n is the link numbers.

$[J]_i$ is the velocity Jacobian matrix of origin of LCS located at joint i + 1,

$\dot{\underline{q}}$ is the joint velocity at current position.

$[J]_i$ is calculated by the methods in Koren's book as described at page 38 of Chapter 2. It is clear that the time increment is changed when $[J]_i$ and $\dot{q}$ are changed. For each origin, one δt_i value is calculated. The smallest δt_i is chosen as the time increment of simulation for the next step. The detailed procedures of this algorithm are shown in the flowchart of this algorithm in Figure 15 on page 81..

4.2.2.2 Intermediate configuration generation algorithms for CP operations

As discussed previously, for CP type operations, the intermediate trajectories are assumed to exactly follow the specified trajectories. A sequence of intermediate points which are generated by interpolating the path determine the intermediate configurations. The resolution is the movement(length between two consecutive intermediate points) along the path(δl) which is determined by the same principle as mentioned above, i.e. $\delta l_E \leq \mathbf{MAX}\{\mathbf{MIN}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{\min,E}\}$, where $D_{\min,E}$ is the distance between the nearest obstacle and the end effector, and each increment at each origin of all LCSs should be less than the corresponding requirement $\delta l_i \leq \mathbf{MAX}\{\mathbf{MIN}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{\min,i}\}$, where $D_{\min,i}$ is the distance between the nearest obstacle and corresponding link i. The procedure of the intermediate configuration generation algorithm is shown in the flowchart of Figure 16 on page 82.

The algorithm starts from setting the end effector's movement to be δl_E which is the maximum allowable increment of end effector. The intermediate point on the specified path, which has a distance of δl_E from the current position, is calculated. Either the intermediate orientation at the trial position is specified or the intermediate orientation at the trial position is calculated under the assumption that the orientation is smoothly changed to the destination. With the trial position and orientation of the next step, the user supported inverse kinematics subroutine of simulated robot is applied to give the joint values of manipulator at that position. Homogeneous transformation is then applied to construct the skeleton of the manipulator configuration of the trial joint values.

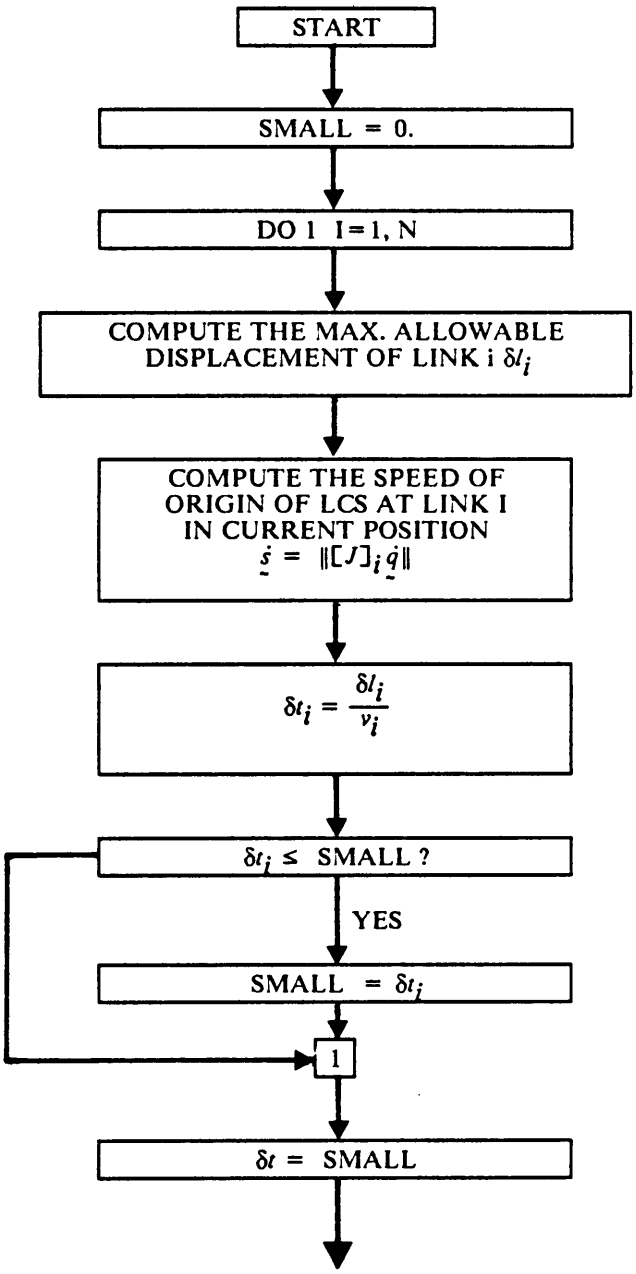


Figure 15. The flowchart of resolution determining algorithm for PTP type operations

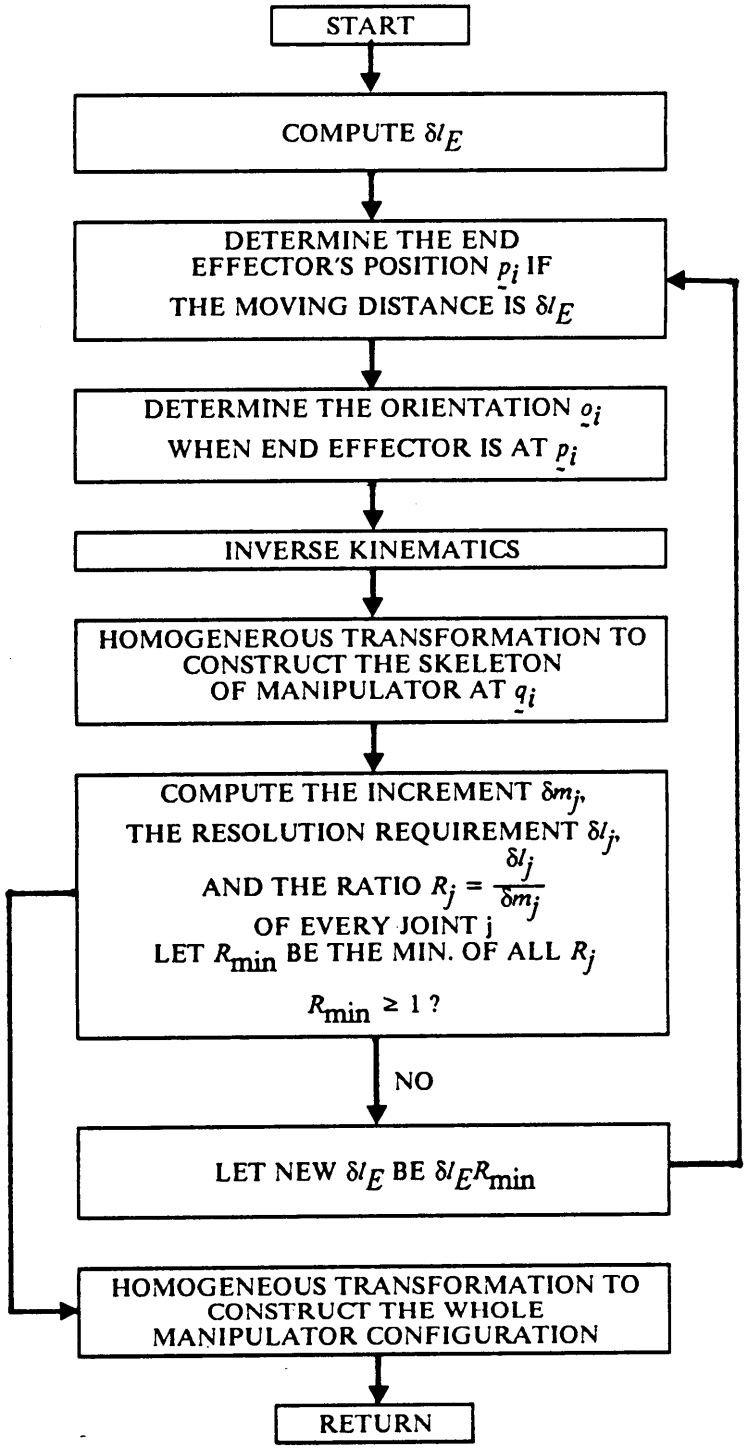


Figure 16. The flowchart of intermediate configuration generation for CP type operations

The ratios of the movement of origin(δm_i) and the maximum allowable movement(δl_i) of every origins of LCS are calculated. If the minimum ratio(R_{min}) is greater than 1, the resolution requirement is met and this trial manipulator configuration is the next desired intermediate configuration. Otherwise, the increment of end effector(δl_E) is factored by the ratio R_{min} and iterate the above procedures until the resolution requirement is met.

Besides the big picture of the algorithm mentioned above, two considerations need to be addressed in detail: the calculation of the intermediate points along the path given the position increment and the problems arising from inverse kinematics.

A. Calculation of the intermediate points for a straight line

For CP operations, GRPVSS can simulate the straight line motion. The intermediate points are determined by interpolating the path. For straight line trajectory, the user needs to specify the position and orientation of the end effector at the starting and ending points. Since the mathematical functions of straight lines are linear, the intermediate points can be easily calculated for a given movement(l).

$$\underline{P}_l = \underline{P}_s + (\underline{P}_f - \underline{P}_s)\frac{l}{L} \qquad (4-17)$$

where

l is the movement from the starting point.

$L = \|\underline{P}_f - \underline{P}_s\|$ is the total length of the path.

$\underline{P}_i$ is the intermediate point when the movement is l.

$\underline{P}_s$ is the starting point of this motion segment.

$\underline{P}_f$ is the ending point of this motion segment.

The orientation is assumed to change smoothly from the starting position to the ending position. That means the change will be proportional to the travel distance.

$$\underset{\sim}{o}_l = Rot(\underset{\sim}{K}, \theta(\frac{l}{L})) \cdot \underset{\sim}{o}_s \qquad (4-18)$$

$$\underset{\sim}{o}_f = Rot(\underset{\sim}{K}, \theta) \cdot \underset{\sim}{o}_s$$

where

$\underset{\sim}{o}_l$ is the intermediate orientation when displacement is l.

$\underset{\sim}{o}_s$ is the starting orientation of this motion segment.

$\underset{\sim}{o}_f$ is the ending orientation of this motion segment.

$\underset{\sim}{K}$ is the axis aout which the orientation of end effector changes from $\underset{\sim}{o}_s$ to $\underset{\sim}{o}_f$

θ is the rotating angle from $\underset{\sim}{o}_s$ to $\underset{\sim}{o}_f$ about the $\underset{\sim}{K}$ axis.

The following is an illustration of why GRPVSS performs a space curve by performing a sequence of straight-line segments instead of tracing the space curve denoted by mathematical functions. The intermediate points, used to determine the intermediate configurations, are obtained by interpolating the specified curve. When a space curve is specified by the mathematical functions, the computation of interpolating the path can be complicated. If the user implements a subroutine which offers the position vectors and orientation vectors given the displacement from the starting point, the computation of intermediate points is easy. Given the displacement, the subroutine is called and the position and orientation of the desired intermediate point are obtained. For an irregular path, however, generating a function which describes the relationship between $\underset{\sim}{s}$ and displacement along the path is very complicated. If the user supports a subroutine that offers the y and z coordinates given the x coordinate, the computation is much more complicated. For one x coordinate, it is possible that there are multi-solutions for the y and z coordinates. For some curves, it is hard to have a general method to judge which pairs are correct. Besides, it would be easy for users to generate a sequence of intermediate points from the mathematical functions of the desired space

curves. Therefore, users need to separate a space curve into a sequence of straight-line motion segments and GRPVSS simulates the specified curve by simulating a sequence of straight-line motion segments.

B. Inverse kinematics subroutine

It is known that multiple manipulator configurations can share the same position and orientation of the end effector. For example, an imaginary plane robot with 3 revolute joints can reach the same state with different configurations as shown in Figure 17 on page 86. Therefore, the inverse kinematics could generate several solutions for one $\underline{s}$. In most real systems, a robot control system chooses one solution as the desired next position based on the given criteria, such as minimizing energy cost, minimizing the execution time, etc. The user supplied inverse kinematics subroutine needs to choose and return one solution according to certain criteria of the simulated robot.

If a work path is generated improperly, the path can be out of workspace. Under this condition, the result of the inverse kinematics subroutine depends on the capability of subroutine, i.e. error may be detected or a wrong solution may be generated. GRPVSS checks every returned solution whether the joint values are within the joint value limits or not. It should be noted that the simulation procedures of CP operations are performed at each simulation step. There is a chance that an unreachable point is skipped in the simulation intermission. The user should note this deficiency. The major purpose of this simulator is collision detection. It would be insufficient to check path feasibility of CP operations with this simulator.

Two nearby robot configurations in the WCS may have corresponding far apart joint vectors. If these two kinds of configurations are successive along the trajectory, the manipulator will move wildly as it passes this discontinuity. This kind of deficiency may happen in the off-line generated paths because the paths are generated in the space described w.r.t. the WCS. Thus, in this condition, the two manipulator configurations of consecutive simulation steps change enormously. The

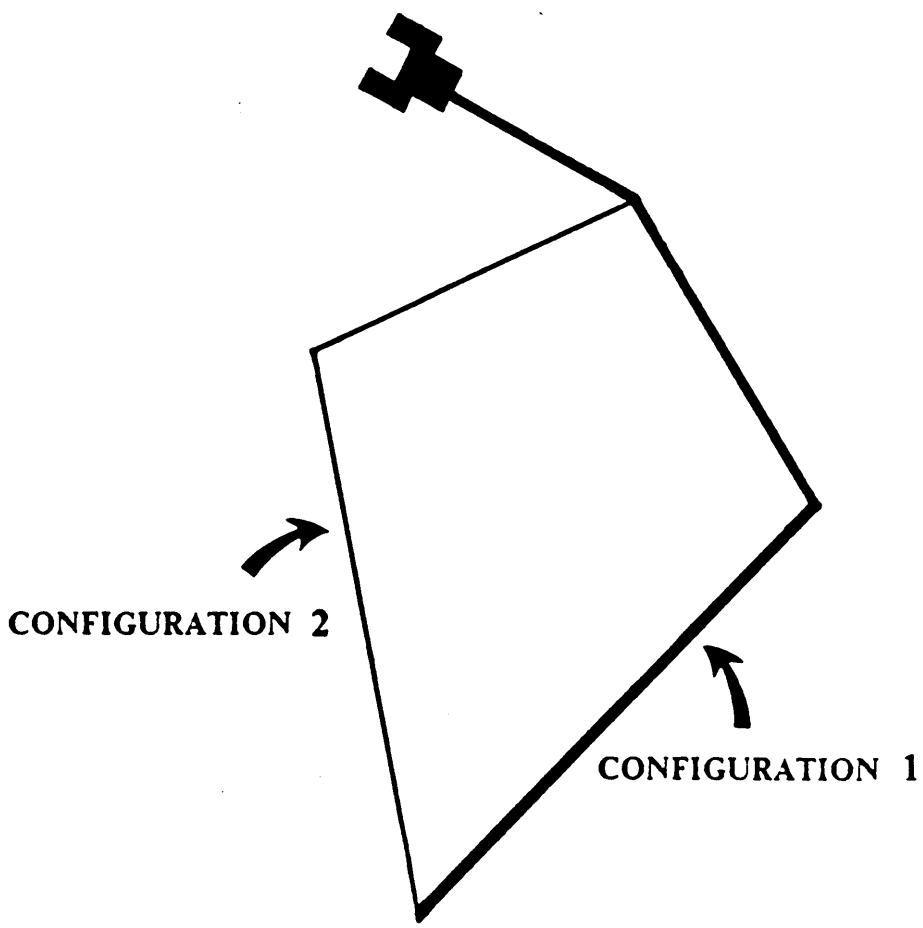


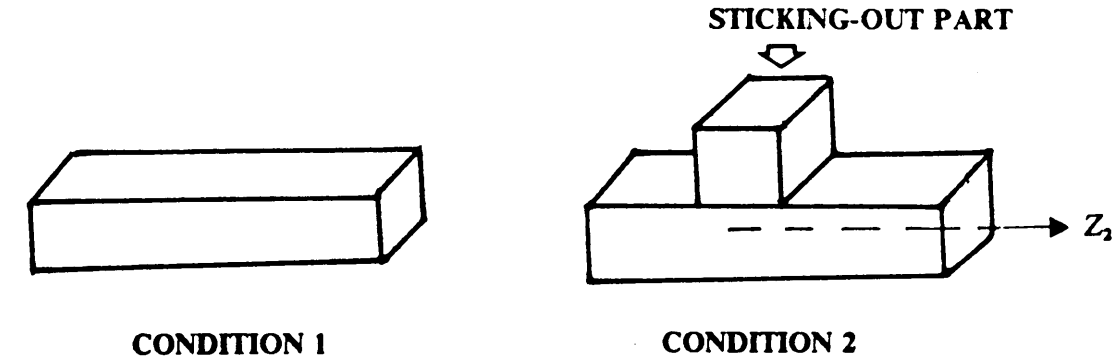
Figure 17. Multiple configurations of an imaginary robot reaching the same position and orienatation of end effector.

program won't detect this path deficiency. However, the user can notice this from the graphics display and the path is regarded as unqualified.

It also needs to be noted that the the maximum allowable increment of each simulation step for both PTP and CP operations is determined by the displacements of the tips of link skeleton. Since the tip of link moves an amount, the rest of this link may move more or less. This method guarantees that collision on each link tip can be 100% detected. If the increment of each simulation step is determined by the displacement of every vertex of the convex hulls which form the robot moving links, 100% collision detection is fully guaranteed. In that case, the above algorithms should be extended to include that the increment at every vertex is less than the corresponding δl value. For PTP operations, the method to calculate the velocity jacobian should be changed to Paul et al's method as described at page 40 to calculate the jacobian of every vertex. Obviously, the computation time of generating the intermediate trajectory will increase for at least $n \times M$ times, where n is the number of vertices of a convex hull, M is the number of convex hulls which form the robot's moving links. GRPVSS consider only the link tips to determine the increment of each simulation step in order to save the computation time, and it still can fit for most industrial robots. In the next paragraph, the points of the links which may have the biggest displacement are discussed under different conditions.

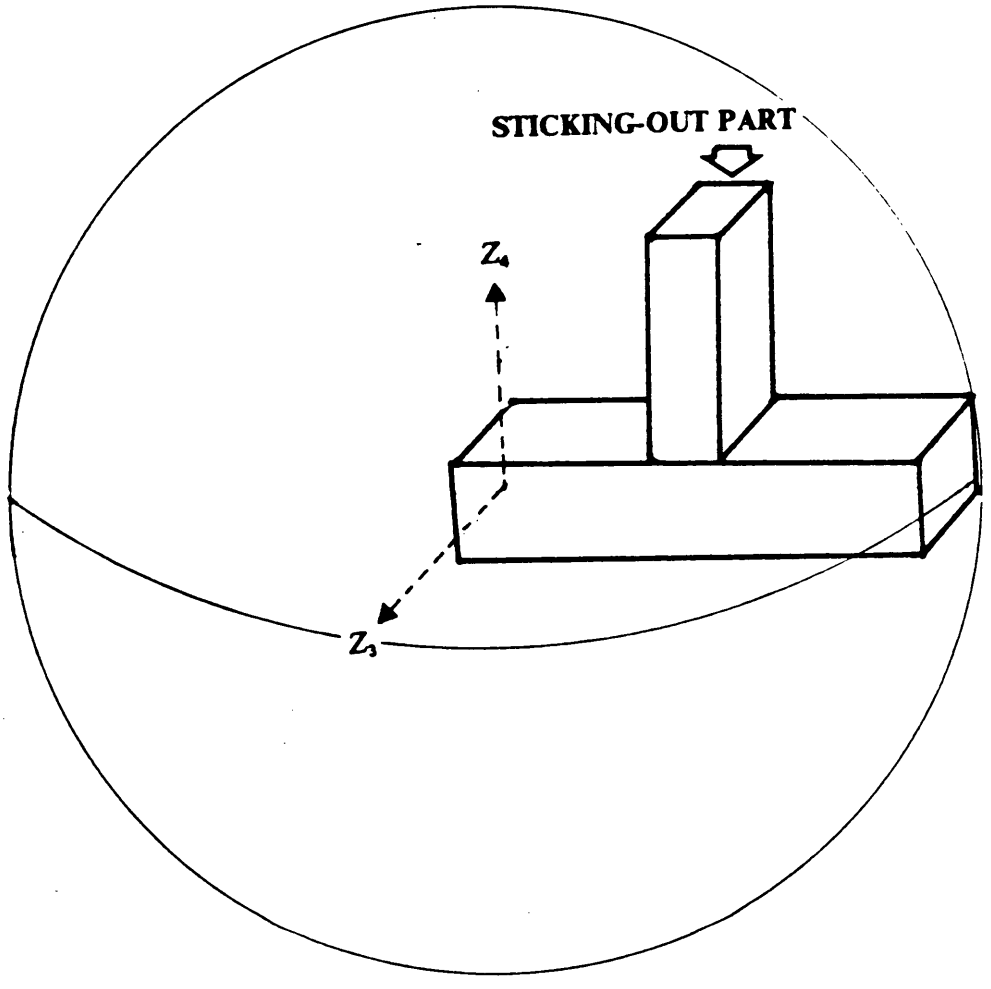
For rigid manipulator, the most critical parts that may have bigger displacement(velocity) than the link skeleton tips are the parts which stick out from the link skeleton. In the following four conditions (as shown in Figure 18 on page 88), the possible locations which have the maximum velocities of the links are described.

1. Robots with only straight-tube links along its skeleton always has the maximum velocity at the end tips. (Most industrial robots have this shape.)
2. If the rotating axis coincides with skeleton the link, such as link 4 of IBM 7545 robot, the maximum velocity always occurs on the sticking-out part.



CONDITION 1

CONDITION 2



CONDITION 3 & CONDITION 4

Figure 18. Four conditions of discussing the locations which have maximum displacements on the robot links.

3. If the the sticking-out part of a link is within the sphere which is formed by using the corresponding link skeleton as the radius and the rotating axis of the link (Z_3 axis in Figure 18) is perpendicular to the sticking-out parts, the maximum velocities may occur on the end tips of the link skeletons or on the tips of sticking-out parts.
4. If the the sticking-out part of robot link is within the sphere which is formed by using the corresponding link skeleton as the radius and the sticking-out parts are parallel to the rotating axis (Z_4 in Figure 18) of that link (such as the link 1 and link 2 of IBM 7545 robot), the maximum velocities are occurred on the end tips of the link skeletons.

For the second condition stated above, the user may avoid it by properly modelling that link into the a straight tube because the sticking-out parts usually do not protrude significantly in that case. For the third case, it is seldom found in industrial robots. Therefore, the current GRPVSS is applicable for most industrial robots.

4.2.3 Interference detection

As discussed in the previous section, collision detection for this simulator is geometric interference between convex polyhedra of manipulator and obstacles. The procedure for collision detection includes two stages: pre-judgement and interference detection. They are described separately in the following paragraphs.

Pre-judge stage

The pre-judgement described in the last section is employed to screen out highly-separated hulls and to save the computing time. The centroid of convex hull is the average of vertices.

$$\underline{P}_{cj} = [x_{cj}, y_{cj}, z_{cj}]^t = [\frac{\sum_{i=1}^{16} x_{ij}}{16}, \frac{\sum_{i=1}^{16} y_{ij}}{16}, \frac{\sum_{i=1}^{16} z_{ij}}{16}]^t \qquad (4-19)$$

where

$\underline{P}_{cj} = [x_{cj}, y_{cj}, z_{cj}]^t$ is the centroid of convex hull j.

$\underline{P}_{ij} = [x_{ij}, y_{ij}, z_{ij}]^t$ is the vertex i of convex hull j.

and the maximum distance between centroid and vertices is:

$$D_j = \max_{1 \to 16} \left(\sqrt{(x_{cj} - x_{ij})^2 + (y_{cj} - y_{ij})^2 + (z_{cj} - z_{ij})^2} \right) \qquad (4-20)$$

The distance between centroids of two convex hulls is:

$$DIS_j = \sqrt{(x_{ci} - x_{cj})^2 + (y_{ci} - y_{cj})^2 + (z_{ci} - z_{cj})} \qquad (4-21)$$

if

$$DIS_{ij} \geq D_i + D_j + \Delta \qquad (4-22)$$

where Δ is the user specified tolerance.

then convex i and convex j should be collision free.

The centroid calculation method may not give an accurate centroid representing the center of the surface. It will cause the D_i to increase. Therefore, this method is quite conservative. The only purpose of this stage is to eliminate the unnecessary interference detection on highly separated hulls. Simple calculation with conservative results meets the purpose of this stage.

Interference detection between two convex hulls

Interference detection between two convex hulls(*i* and *j*) is performed between the edges of one hull *i* and the surfaces of the other hull *j*. The checking is also performed between all edges of hull *j* and all surfaces of hull *i*. The flowchart of interference detection is shown in Figure 19 on page 91. The algorithm of interference detection between a line segment and a surface is described as follows:

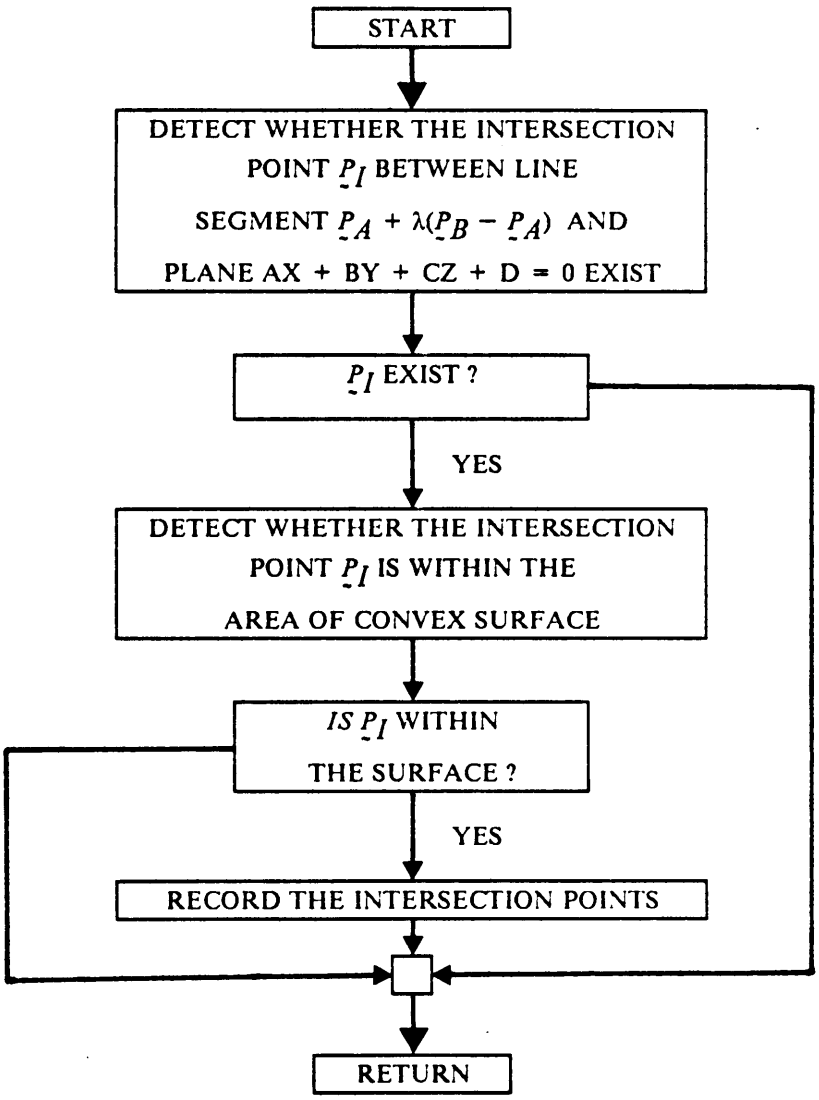


Figure 19. The flowchart of interference detection algorithm.

1. Generating the plane equation of the surfaces:

   The plane equation, $Ax + By + Cz + D = 0$, of surface with vertices $\underline{P}_1(x_1, y_1, z_1)$, $\underline{P}_2(x_2, y_2, z_2)$, and $\underline{P}_3(x_3, y_3, z_3)$. can be derived as:

$$\det \begin{bmatrix} x - x1 & y - y1 & z - z1 \\ x2 - x1 & y2 - y1 & z2 - z1 \\ x3 - x1 & y3 - y1 & z3 - z1 \end{bmatrix} = 0 \qquad (4-23)$$

   Then, the corresponding parameters A, B, C, D, can be obtained by calculating the determinant of the matrix of Eq.(4-23).

2. Computing the intersection point $\underline{P}_I$ of the line segment $\underline{P}_a + \lambda (\underline{P}_a - \underline{P}_b)$, $0. \le \lambda \le 1.$ and the plane $Ax + By + Cz + D = 0$, that contains the surface.

$$\underline{P}_I = \left[\underline{P}_a, \underline{P}_b\right] \left[\lambda_1, 1 - \lambda_1\right]^t \qquad (4-24)$$

   where $\lambda_1 = \dfrac{-(\underline{P}_a^t \cdot \underline{N} + D)}{\underline{N}\left[\underline{P}_b - \underline{P}_a\right]^t}$.

   If $0. \le \lambda_1 \le 1.$, the line segment intersects with the plane. Then the procedure goes to step 3; Otherwise, the line segment doesn't intersect with the plane and no interference is concluded.

3. As shown in Figure 20 on page 93. If

$$\left[(\underline{P}_{I+1} - \underline{P}_I) \times (\underline{P}_{I+1} - \underline{P}_I)\right] \cdot \left[(\underline{P}_{I+1} - \underline{P}_i) \times (\underline{P}_{I+2} - \underline{P}_{i+1})\right] > 0 \qquad (4-25)$$

   for all vertices *i* of the surface, $\underline{P}_I$ is within the surface. That is, interference does exist and $\underline{P}_I$ is the intersection points. Otherwise, no interference is concluded.

In this chapter, the structure of GRPVSS has been described in Section 4.1. The simulation procedures of GRPVSS is hierarchical described according to the flowchart. The major algorithms of

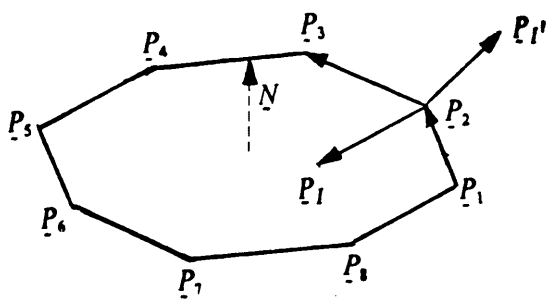


Figure 20. Determining whether or not a point is inside a surface by using vector cross product.

GRPVSS: robot expansion algorithm, intermediate configuration generation algorithms, and interference detection algorithms have been discussed in Section 4.2. In the next chapter, this thesis is concluded with the summary of all methods and algorithms of GRPVSS and discussion about computation complexity, tolerance value determination, and further extension and application of GRPVSS.

Chapter 5 CONCLUSION and DISCUSSION

This research has created a general robot simulation system for path collision verification using algorithms that determine the maximum robot simulation resolution without causing undetected collisions. In this chapter, a summary of this thesis is presented in Section 5.1; then discussions about computation complexity of the generated algorithms, determination of tolerance value, further extension and applications of GRPVSS are presented in Section 5.2.

5.1 Conclusion

This thesis presents a comprehensive discussion on generating a General Robot Path Verification Simulation System, GRPVSS. The general background of robotics and literature review have been presented in Chapter 1 and Chapter 2 respectively. Chapter 1 introduces the general background of robotics which is needed for this research, defines some of the terminologies, and discusses the need of off-line path verification. Chapter 2 reviews major articles relevant to this research, particularly in the areas of off-line path planning, robot kinematic simulators, robot dynamic simulators, collision detection, and geometric interference detection. From the general background and

literature review, the objective of research is defined in Chapter 3. The research objective is to explore a general purpose robot path verification simulation system which has the following characteristics: generality, simplicity, 100% collision detection, and close real trajectory tracing. The methodology and algorithms of GRPVSS to achieve the objective have been described in Chapter 3 and Chapter 4. In the following paragraphs, the methodologies and algorithms used to achieve the objective are summarized.

In order to be applicable for all open-looped industrial robots, robots in GRPVSS are modelled by kinematic parameters, θ, a, α, d, of all links according to DH convention, and direct kinematic transformation constructs the manipulator's skeleton. Shape parameters are used to describe the space occupied by the manipulator and obstacles. Instead of trying to catch every detail of the objects' geometric shape, GRPVSS uses a convex polyhedron or a union of convex polyhedra to represent the object they contain. Thus, the shape parameters of the manipulator are the coordinates (described w.r.t. the LCS) of vertices of those convex hulls which contain links. Homogeneous transformation transfers those shape parameters to the coordinates described w.r.t. the WCS and construct the whole body of the manipulator. The shape parameters of obstacles (objects in the working environment) are the coordinates (described w.r.t. WCS) of the vertices of the convex hulls which contain the obstacles. This approximation with convex hulls substantially reduces the computation of collision detection, and the resulting algorithm can be generally applied on any manipulator. Both simplicity and generality are achieved by this approach.

Industrial robots move their arms under PTP or CP commands. PTP commands move the robot from its starting state to destination state, either described w.r.t. the joint values($\underline{q}_s$, $\underline{q}_d$) or described w.r.t. the WCS($\underline{s}_s$, $\underline{s}_d$) without specifying the path between the starting point and ending point. CP commands specify starting and ending position and orientation of the end effector described w.r.t. the WCS($\underline{s}_s$, $\underline{s}_d$), and all joints are controlled to trace the required straight-line trajectory. GRPVSS can simulate both of these operations.

The trajectory tracing is the key issue for collision detection purpose. The real trajectory is determined by the assembly of the robot controller and the dynamics of the manipulator-actuator as well as the load on the end effector. Since dynamic simulation is too time consuming and the dynamic parameters are difficult to identify, this simulator applies a group of motion time profiles to provide information for imitating manipulator trajectory of PTP operations without considering the limited variance caused by the load. For each joint actuator, there is one function to describe the relationship between joint velocity and time based on the velocity feedback control. That is the motion time profile. The ideal case is the ideal trapezoid function as shown in Figure 2 on page 13. The user must provide these motion time profiles in the data file if needed. Otherwise, the ideal functions are applied. For the CP operations, the robot's end effector is assumed to follow exactly a specified straight line segment. Since the purpose of this simulation system is to check for the possible collisions with obstacles that are static, the velocity or time factor is not important. Therefore, the trajectory of CP operations is determined by the specified path (described w.r.t. the WCS) of the end effector. In summary, GRPVSS possesses the simplicity of a kinematic simulator and considers the dynamics which can be included in the motion time profiles. These features greatly increase the precision of trajectory tracing without the deficiencies of a dynamic simulator.

The collision detection between a moving robot and static obstacles is designed to be performed by detecting interference between the robot and any obstacle at each simulation step. In order to have 100% path verification, collision should not occur during the intermission of two consecutive simulations (resolution of simulation). The manipulator's arms are expanded by an amount of user specified tolerance to include the robot inaccuracy and safety factor. If the distribution of the obstacles is not considered, the amount of intrusions of obstacles without collision detected within two consecutive simulations should be less than the user-specified tolerance value as shown in Figure 9 on page 63 and be less than the $2\Delta + L$, where L is the smallest thickness of all obstacles. It turns out that the displacement should be less than $\mathbf{MAX}\{\mathbf{Min}[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{min}\}$. However, if the obstacles is located outside the distance δX, the distance to the nearest obstacle is the maximum allowable displacement. Thus the movement of moving links should not be bigger

than the corresponding $\delta l = \max\{ \min[\frac{2\Delta}{\cos\frac{\theta}{2}}, 2\Delta + L], D_{min}\}$, , where θ is the smallest angle of the obstacle hulls, and D_{min} is distance between the moving link and its nearest obstacle, and L is the smallest thickness of all obstacles. From this observation, the resolution requirements for CP and PTP operations are derived. The derivation is trying to find the maximum simulation intermission without violating the above principle.

According to the trajectory generation mechanisms for PTP and CP operations, the increments for each simulation step are a time factor for PTP operations, and the movement of end effector along the specified path for CP operations. The time increment, δt, for joint i + 1 is $\delta t_i \leq \frac{\delta l_i}{v_i}$, where v_i is the speed of joint i + 1, δl_i is the maximum allowable movement of link i. The smallest δt_i, $i = 1, 2, \ldots, N.$, where N is the link number, is selected as the time increment from the current step to the next step. For a CP operation, the end effector's movement along the path is set to δl_N at first. A recursive algorithm is generated to compute the movement of every joint and proportionate the end effector's movement until all joint's movements are less than their corresponding maximum allowable movements. The δl_i is determined for every joint, i.e. the tips of the links. This method guarantees that collision on each link tip can be 100% detected. From the formulation of δl, it is noticed that a larger tolerance causes a lower resolution level, fewer simulation steps, less computation time, and a more conservative result.

The whole robot arms and obstacles are contained by those convex polyhedra. If collisions occur between the manipulator and obstacles, the interference would also occur between the convex hulls of manipulator links and the convex hulls of obstacles. This approach is conservative because free space is wasted due to the approximation in the modelling. However, complexity is reduced and significant computation time is saved. The computation for interference-detection is proportional to n^2 , where n is the number of vertices of the hulls which are employed to model the robot and environment. A curve edge is considered to contain infinitely many vertices, while its approximation line segment contains only two vertices. This approximation also results in linear intersection function. Therefore, significant computation time is saved. Since the robot and obstacles are

formed by convex polyhedra, interference detection becomes much simpler. If there is interference, at least one edge of one convex hull will interfere with the surface of the other one. Therefore, the only detection needed is the interference between edges and surfaces. In order to save unnecessary detection between two highly-separated convex polyhedra, pre-judgement using distance between centroids is processed.

GRPVSS has been developed on the IBM VM/CMS system with Fortran 77. Test result indicates that GRPVSS does satisfy the design objectives discussed in Scetion 3.1. The execution time is quite acceptable for practical usage (This point will be discussed in the next section). The user's guide of GRPVSS is in Appendix A. Several graphics displays are also included in Appendix A.

In summary, this research creates a robot simulation system with 100% path verification capability. Motion time profiles are included in the kinematic simulator to increase the precision of trajectory tracing. It also develops a method to determine the maximum simulation resolution of robot path verification simulation to have 100% collision detection guarantee by performing interference detection at each simulation step. This work extends Ahuja et al's proposal of a collision detection algorithm[1] and apply the extended algorithms for collision-free path verification purpose.

5.2 Discussion

In this section, computation complexities of the major algorithms, tolerance value determination, and further extensions and applications of GRPVSS are discussed in the following subsections individually.

5.2.1 Computation complexity of major algorithms

The calculation numbers of the convex polyhedron expansion algorithm, the interference detection algorithm, and the intermediate configuration generation for PTP operations are shown in Table 2.

Table 2. The calculation numbers of major algorithms of GRPVSS

Algorithm	Additions	Multiples
Expanding one convex hull of n vertices	$72n + 22$	$65n + 18$
Interference Detection between two convex polyhedra of n vertices	$276(\frac{n}{2})^2 + 48(\frac{n}{2})$	$183(\frac{n}{2})^2 + 42(\frac{n}{2})$
Intermediate configuration generation for PTP operations of N link robot	$3N^2 + 18N + 3$	$2N^2 + 30N + 9$

The computation complexity of the intermediate configuration generation algorithm for CP operations depends on the computation complexity of the subroutine which solves the inverse kinematics problems of the simulated robot. The calculation number of the inverse kinematics subroutine can not be estimated. Therefore, the calculation number of intermediate configuration generation for CP operations is not listed in Table 2.

The calculation numbers are counted in the worst case. For example, the interference detection procedure can be terminated at the pre-check stage, or at the step 1 or step 2 of interference detection procedures without completing the all procedures. The calculation number of the PTP trajectory generation mechanism for each configuration are based on the ideal trapezoid profiles. If motion-time profiles of the simulated robot are applied, the calculation number could be more.

The execution time of of simulating the IBM 7545 robot is shown in the next paragraph as an example.

Example

Table 3. The execution time of major algorithms of GRPVSS(on IBM 4341)

Algorithm	Execution time (seconds)
Expanding one convex hull of 16 vertices	0.016
Interference detection between two overlapping convex polyhedra of 16 vertices	0.202
Intermediate configuration generation for PTP operations	0.012
Intermediate configuration generation for CP operations	0.04
A PTP operation of IBM 7545 robot	11.68
A CP operation of IBM 7545 robot	5.70

If the convex hull is set to have 16 vertices and the robot is the IBM 7545 robot which has 4 links, the average execution times of several procedures on the IBM 4341 mainframe computer at Virginia Tech's VTVM3 mainframe computer system are shown in Table 3. Two obstacles are in the robot working space. The tolerance is set to be 0.5 inches and the desired joint velocities are 3° per second for revolute joints and 0.3 inches per second for the prismatic joint. For each path, one obstacle is on the path and the other is not. Expanding one convex hull of 16 vertices takes 0.016 seconds. Interference detection between two overlapping convex hulls needs 0.202 seconds. The execution time needed for generating the manipulator's intermediate configuration of each simulation step is 0.012 seconds for PTP operations and 0.04 seconds for CP operations. For a PTP operation of 50° increment in revolute joints, 4.0 inch increment in the prismatic joint, the execution time is 11.68 seconds. For a CP operation of a 32.57 inch straight line, the whole execution time is 5.70 seconds. From these observations, it clearly shows that the computation time is quite acceptable for practical usage.

5.2.2 Tolerance value

It has been discussed in the previous sections that the tolerance is an important factor of resolution. The bigger tolerance is specified, the bigger simulation resolution can be used, and the more conservative result is obtained, but less computation time is needed. It is noted that the tolerance value should be bigger than the positioning accuracy and repeatability that the robot manufacturer claims. The method for measuring the robot's tracking errors can be referenced on Wu and Eman's paper[80]. On the other hand, large tolerance value can cause conservative result and lose in accuracy during collision detection. Therefore, the user is suggested to choose an appropriately large tolerance value to simulate the path. If the result shows collision free, it is guaranteed that there is no collision in this path. If collision is detected, the possibility of non-collision still exist because of the big tolerance value. The user may utilize the option of GRPVSS to change the tolerance to the magnitude of the positioning accuracy. If the result still shows collision occurred, this path is not qualified.

5.2.3 Further Extension

Current GRPVSS is modelled to simulate one robot working in a static environment (unmoving objects in the workspace). This simulation system can be extended to simulate multiple robots working in dynamic environments. The collision detection algorithm does not need to be modified. The resolution determination for PTP operation needs to change the velocity in Eq.(4-14), which determines the time increment, to the relative velocity of the moving robots and moving obstacles. The path of the moving obstacles needs to be modelled. Big modification may be required in CP operations. Since the current CP operation does not consider the velocity or the time factor, the methodology of trajectory generation for CP operation needs to be changed. Further investigation

should focus on methods which include the control system characteristics of the robots. Since robots and obstacles are in motion, the configurations used to detect interferences should be the corresponding configurations of robots and obstacles at a certain time. That means, the trajectory should be determined in the time domain. Dynamic simulation probably can be used. However, the increased computation time is the necessary consequence.

In the program of GRPVSS, the graphical animation part is basically separated from the main structure. It will not be difficult to do the modifications for application on other graphical software. Furthermore, it is believed that GRPVSS can be modified to be applied on a personal computer(PC). A hard-disk may be needed on the regular IBM PC for enough memory space. The execution time will be enormously increased, and the graphics resolution could be lower. However, the applicability will be increased because the cost on a PC is much lower than on the IBM 4341 mainframe computer system. Besides, the computation speed and the memory space of the new personal computers are rapidly increasing.

5.2.4 Application of GRPVSS

Industrial robots have been widely used in industry. Path planning is one of the most frequent tasks in the plants which employ robots. On every plant floor on which robots work, there are obstacles for robots. Collision-free path is definitely a prerequisite for any working path. When on-line manual teaching is performed to plan a robot path, the operator takes care of the collision avoidance. However, on-line manual teaching is expensive because equipment is tied up. Off-line path planning is the goal of robot manufacturers and users. For off-line path planning, an off-line path verification facility is needed to verify that the generated paths are collision-free. GRPVSS, which is capable of 100% collision detection, has strong capability to do the path verification and is quite applicable in industry. A robot operator can use GRPVSS to verify the off-line generated path

before running the path on a real system. Even for those plants utilizing the manual teaching method to generate working paths, the operator can use GRPVSS to check the feasibility of putting some objects into the working environment and can quickly obtain a conservative answer without breaking down the machine schedule and can reduce the on-line planning time. Besides, GRPVSS's features such as interactive communication with the user, good accuracy, nice graphics animation, and acceptable execution time, makes it a promising tool for robot operators in industry.

REFERENCE

1. Ahuja, N., R. T. Chien, R. Yen, and N. Bridwell, "Interference Detection and Collision Avoidance Among Three Dimensional Objects," Proceeding of First Annual National Conference on Artifical Intelligence, Aug. 1980, pp. 44-48.
2. Ahuja, N. and C. Nash, "Octree Representation of Moving Objects," Computer Vision, Graphics, and Image Processing, Vol. 26, 1984, pp.207-216.
3. Baer, A., C. Eastman, and M. Henrion, "Geometric Modelling: A Survey." Computer-Aided Design, Vol. 11, 1979, pp.253-272.
4. Bobrow, J. E., S. Dubowsky, and J. S. Gibson, "On the Optimal Control of Robot Manipulatos with Actuator Constraints," Proceeding of 1983 American Control Conference, San Fancisco, CA, pp.782-787.
5. Bonney, M. C., M. Dooner, N. K. Taylor, J. L.Green, and W. B. Heginbotham, "Verifying Robot Programs for Collision Free Tasks," Developments in Robotics, 1983, pp. 257-263.
6. Boyse, J. W. "Interference Detection Among Solids and Surfaces," Communication of ACM, Vol. 22, No. 1, January 1979, pp. 3-9.
7. Brooks, R. A., "Symbolic Reasoning Among 3-D Models and 2-D Images," Artifical Intelligence Vol. 17, 1981, pp. 285-348.

8. Brooks, R. A., "Planning Collision-Free Motions for Pick-and-Place Operations," The International Journal of Robotics Research, Vol. 2, No. 4, Winter, 1983, pp. 19-44.

9. _____, "Solving the Find-Path Problem by Good Representation of Free Space," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-13, No. 3, March/April 1983, pp. 190-197.

10. Brady, M., Hollerbach, J. M., Johnson, T. L., Lezano-Perez, T., and M. T. Mason, Robot Motion: Planning and Control, MIT Press, 1982.

11. Buckley C. E. and L. J. Leifer, "A Proximity Metric for Continuum Path Planning," Proceeding of Ninth International Conference on Artifical Intelligence, 1985, pp. 1096-1102.

12. Canny, J. "A Voronoi Method for the Piano-Movers Problems," Proceeding of IEEE International Conference on Robotics and Automation, March 1985, pp. 530-535.

13. Canny, J. "Collision Detection for Moving Polyhedra," IEEE Transactions on Pattern Analysis And Machine Intelligence, Vol. PAMI-8, No. 2, March 1986, pp. 200-209.

14. Chien, C. H. and J. K. Aggarwal, "A Volume/Surface Octree Representation," Proceeding of 7th International Conference on Pattern Recognition, 1984, pp.817-820.

15. Comba, P. G., "A Procedure for Detecting Intersections of Three-Dimensional Objects," Journal of ACM, Vol. 15, No. 3, 1968, pp. 354-366.

16. Denavit, J., and R. S. Hartenberg, "A Kinematic Notation for Lower Pair Mechanisms Based on Matrices", Journal of Applied Mechanics, ASME, Vol. 22, June 1955, pp. 215-221.

17. Derby, S., "Simulation Motion Elements of General-Purpose Robot Arms," The International Journal of Robotics Research, Vol. 2, No. 1, Spring 1983, pp. 3-12.

18. Dubowsky, S. and D. T. DesForges, "The Application of Model-Referenced Adaptive Control to Robotic Manipulators," Journal of Dynamic Systems and Measurement Control, Vol. 101, 1979, pp. 193-200.

19. Engleberger, J. F., Robotics in Practices: Management and Applications of Industrial Robots, Amacon, 1980.

20. Falb, P. L. and W. A. Wolovich, "Decoupling in the Design and Synthesis of Multivariable Control Systems," IEEE Transactions on Automatic Control, Vol. 12, No. 6, 1967, pp.651-655.

21. Freund, E. "Fast Nonlinear Control with Arbitrary Pole-Placement for Industrial Robots and Manipulators," International Journal of Robotics besearch, Vol. 1, No. 1, 1982, pp. 65-78.

22. Fu, K. S., R. C. Gonzalez, and C. S. G. Lee, Robotics: Control, Sensing, Vision and Intelligence, 1987, McGraw-Hill, New York.

23. Gilbert, E. G. and I. J. Ha, "An Approach to Nonlinear Feedback Control with Application to Robotics," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-14, No. 2, 1984, pp.101-109.

24. Gilbert, E. G. and D. W. Johnson, "Distance Functions and Their Application to Robot Path in the Presence of Obstacle," IEEE Journal of Robotics and Automation, Vol. RA-1, No.1, March 1986, pp. 21-30.

25. Gini, G., M. Gini, R. Gini, and D. Guise, "Introducing Software Systems in Industrial Robots," Proceeding of 9th International Symposium on Industrial Robots, Washington D. C., March 1979, pp. 309-321.

26. Golla, D. F., S. C. Garg, and P. C. Hughes, "Linear State-Feedback Control of Manipulators," Mechanical Machine Theory, Vol. 16, 1981, pp. 93-103.

27. Grossman, D. D., "Robotic Software," Mechanical Engineering, Vol. 104, No. 8, August 1982, pp. 46-47.

28. Hemami, H. and P. C. Camana, "Nonlinear Feedback in Simple Locomotion Systems," IEEE Transactions on Automatic Control, Vol. AC-19, 1976, pp.855-860.

29. Herman, M., "Fast, Three-Dimensional, Collision-Free Motion Planning," Proceeding of IEEE International Conference on Robotics and Automation, 1986, pp. 1056-1063.

30. Itkis, U., Control system of variable structure, 1976, John Wiley, New York.

31. Jackins, C. L. and S. L. Tanimoto, "Oct-tree and Their Use in Representing Three-Dimensional Objects," Computer Graphics and Image Processing, Vol. 19, 1982, pp. 129-147.

32. Jackson, C. L. and S. L. Tanimoto "Oct-Tree and Their Use in Representing Three-Dimensional Objects," Computer Graphics and Image Processing, Vol. 14, 1980, pp. 249-270.

33. Kahn, M. E., and B. Roth, "The Near-Minimum-Time Control of Open-loop Articulated Kinematic Chains," ASME Transaction on Dynamic Systems, Measurement and Control, Vol. 93, 1971, pp.164-172.

34. Kambhampati, S. and L. R. Davis, "Multiresolution Path Planning for Mobil Robots," IEEE Journal of Robotics and Automation, Vol. RA-2, No.3, Sep. 1986, pp. 135-145.

35. Khatib, O. "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," The International Journal of robotics Research, Vol. 5, No. 1 , Spring 1986, pp. 90-98.

36. Kitajima, K., "Robot Simulation Based On a General-Purpose Structure Model for Machine," Proceeding of IEEE International Conference on Robotics and Automation, 1987, pp. 1909-1915.

37. Koren, Y., Robotics for Engineers, Mcgraw-Hill, New York, 1985.

38. Koivo, A. J. and T. H. Guo, "Adaptive Linear Controller for Robotic Manipulators," IEEE Transactions on Automatic Control, Vol. AC-28, No. 1, 1983, pp.162-171.

39. Lee, C. S. G., and M. J. Chung, "An Adaptive Control Strategy for Mechanical Manipulators," IEEE Transactions on Automatic Control, Vol. AC-29, No. 9, 1984, pp.837-840.

40. Lee, C. S. G., and M. J. Chung, "Adaptive Perturbation Control with Feedforward Compensation for Robot Manipulators," Simulation, Vol. 44, No. 3, 1985, pp.127-136.

41. Lee, C. S. G., and B. H. Lee, "Resolved Motion Adaptive Control for Mechanical Manipulators," ASME Transactions on Dynamic Systems, Measurement and Control, Vol. 106, No. 2, 1984, pp.134-142.

42. Lee, C. S. G., T. N. Mudge, J. L. Turney, "Hierarchical Control Structure Using Special Purpose Processors for the control of Robot Arms," Proceeding of 1982 Pattern Recognition and Image Processing Conference, Las Vegas, Nev., pp.634-640.

43. Leu, M. C. and R. Mahajan, "Computer Graphic Simulation of Robot Kinematics and Dynamics," Proceeding of Robot 9 Conference: Applications for Today, Vol. 1, 1984, pp. 4-80 - 4-101.

44. Lieberman, L. I. and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," IBM Journal in Research Development, Vol. 21, No. 4, 1977, pp. 321-333.

45. Lozano-Perez T. "The Design of a Mechanical Assembly System," Artifical Intelligence Lab. MIT. AI TR 397, 1976.

46. Lozano-Perez T. and M. A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles", Communication of ACM, Vol. 22, No. 10, October 1979, pp. 560-570.

47. Lozano-Perez, T. "Automatic Planning of Manipulator Transfer Movements," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 10, Oct., 1981. pp. 681-698.

48. _____, "Spatial Planning: A Configuration Space Approach," IEEE Transactions on Computers, Vol. C-32, No. 2, Feb., 1983, pp. 108-120.

49. Luh, J. Y. S., W. D. Fisher, and R. P. C. Paul, "Joint Torque Control by a Direct Feedback for Industrial Robots," Proceeding of 20th IEEE Conference on Decision and Control, 1981, pp. 265-271.

50. Luh, J. Y. S., M. W. Walker, and R. P. C. Paul, "On-Line Computational Scheme for Mechanical Manipulators," Journal of Dynamics systemsm, Measurement, and Control, Vol. 102, 1980, pp. 69-76.

51. Luh, J. Y. S., M. W. Walker, and R. P. C. Paul, "Resolved Acceleration Control of Mechanical Manipulators," IEEE Transaction on Automatic Control, Vol. 25, No. 3, 1980, pp. 468-474.

52. Luh J. Y. S. and C. E. Campbell, "Collision-Free Path Planning for Industrial Robots," Proceeding of 21st IEEE Conference on Decision Control, 1982, pp. 84-88.

53. Lumelsky, V. J., "Continuous Motion Planning in Unknown Environment for a 3D Cartesian Robot Arm," Proceeding of IEEE International Conference on Robotics and Automation, 1986, pp. 1050-1055.

54. Maruyama, K. "A Procedure to Determine Intersections Between Polyhedral Objects," International Journal of Computer and Information Sciences, Vol. 1, No. 3, 1972, pp. 255-266.

55. Meagher, D. "Geometric Modelling Using Octree Encoding," Computer Graphics and Image Processing, Vol. 19, 1982, pp. 129-147.

56. Milenkovic, V. and B. Huang, "Kinematics of Major Robot Linkages," Proceeding of 13th International Symposium on Industrial Robots, Chicago, pp.16-31 to 16-47.

57. Nemec B. and J. Lenarcic, "A Robot Simulation System Based on Kinematic Analysis," Robotica, Vol. 3, pp. 79-84.

58. Nilsson, N. Principles of Artifical Intelligence, Tigoa Publishing, CA. 1980.

59. Paul, B. and J. Rosa, "Kinematics Simulation of Serial Manipulator," The International Journal of Robotics Research, Vol. 5, No. 2, Summer 1986, pp. 14-31.

60. Paul, Richard P., "Manipulator Cartesian Path Control," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-9, 1979, pp. 702-711.

61. Paul, Richard P., Robot Manipulators: Mathmatics, Programming, and Control, MIT press, 1981.

62. Pieper, D. C., "The Kinematics of Manipulators Under Computer Control," ARPA order No. 957, Stanford University, Stanford, CA, 1968.

63. Requicha, A. A. G. "Representation for Rigid Solids: Theory, Methods, and Systems," Computing surveys, Vol. 12, No. 4, 1980, pp.437-464.

64. Ruff, R. and N. Ahuja, "Path Planning in a Three Dimensional Environment," Proceeding of Seventh International Conference on Pattern Recognition, Canada, 1984, pp. 188-191.

65. Saridis, G. N. and C. S. G. Lee, "An Approximation Theory of Optimal Control for Trainable Manipulators," IEEE Transactions on Systems, Man, Cybernetics, Vol. SMC-9, No. 3, 1979, pp.152-159.

66. Shamos, M. I., Computational Geometry, Springer-Verlag, New York, 1977.

67. Sheth, P. N. and J. J., Jr. Uicker, "A Generalized Symbolic Notation for Mechanisms," Journal of Engineering for Industry, Transaction, ASME, Series B, 1971, pp. 93:102-112.

68. Srihari, S. N. "Representation of Three-Dimensional Digital Images," Computering Surveys Vol. 13, No. 4, 1981, pp. 399-424.

69. Sungurtekin U. A. and H. B. Voelcker, "Graphical Simulation & Automatic Verification of NC Machine Programs," Proceeding of IEEE International Conference on Robotics and Automation, 1986, pp. 156-165.

70. Takegaki, M. and S. Arimoto, "A New Feedback Method for Dynamic Control of Manipulators," Journal of Dynamic Systems, Measurement, and Control, Vol. 102, 1981, pp.119-125.

71. Taylor, R. H., "Planning and Execution of Straight-Line Manipulator Trajectories," IBM Journal of Research and Development, Vol. 23, 1979, pp.424-436.

72. Tsai, L. and A. Morgan, "Solving the Kinematics of the Most General Six- and Five-degree-of-freedom Manipulators by Continuation Methods," Paper 84-DET-20, ASME Mechanisms Conference, Boston, October 7-10, 1984.

73. Udupa, S. "Collision Detection and Avoidance in Computer Manipulator," Procceding of 5th International Joint Conference on Artifical Intelligence, MIT, 1977. pp.737-

74. Uicker, J. J., Jr., J. Denavit, and R. S. Hartenberg, "An Iterative Methods for the Displacement Analysis of Spatial Mechanism," ASME Transactions on Applied Mechanics, Vol. 31, Series E, 1964, pp.309-314.

75. Utkin, V. I., "Variable Structure Systems with Sliding Mode: A survey," IEEE Transactions on Automatic Control, Vol. AC-22, 1977, pp.212-222.

76. Walker, M. W. and D. E. Orin, "Efficient Dynamic Computer Simulation of Robotic Mechanisms," Journal of Dynamic Systems, Measurement, and Control, Vol. 104, 1982, pp. 205-211.

77. Whitney, D. F., "Resolved Motion Rate Control of Manipulators and Human Prostheses," IEEE transactions on Man-Machine Systems, Vol. MMS-10, 1969, pp. 47-53.

78. Whitney, D. F., "The Mathematics of Coordinated Control of Prostheses Arm and Manipulators," Journal of Dynamic Systems, Measurement, and Control, Dec. 1972, pp. 303-309.

79. Wong, E. K. and K.S. Fu, " A Hierarchical Orthogonal Space Approach to Three-Dimensional Path Planning," IEEE Journal of Robotics and Automation, Vol. RA-2, No.1, March 1986, pp. 42-53.

80. Wu, B. T. and K. F. Eman, "Analysis of Dynamic Tracking Errors of Robots," Proceedings of the IXth ICPR, August, 1987, pp. 1941-1946.

81. Wu, C. H. and R. P. Paul, "Resolved Motion Force Control of Robot Manipulator," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-12, No. 3, pp.266-275.

82. Young, K.-K. D., "Controller Design for an Manipulator Using Theory of Variable Structure Systems," IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-8, No. 2, 1978, pp. 101-109.

Appendix A. USER'S GUIDE for GRPVSS

This thesis reports the construction process of the simulation system, GRPVSS. The structure and major algorithms of GRPVSS have been described in Chapter 4. The applications and possible extension of GRPVSS have also been shown in Chapter 5. As mentioned before, GRPVSS has been implemented on the VM/CMS system on the IBM 4341 mainframe computer at Virginia Tech's VTVM3 mainframe computer system. using FORTRAN 77. The graphical part is completed by using the subroutines of graPHIGS, one of the graphics software published by IBM. Users are recommended to read the previous chapters to understand the characteristics of GRPVSS and its applications. With those understandings, users can obtain the necessary information for running GRPVSS in this guide. Four sections are included in this user's guide:

1. preparation of input,
2. running procedures (including the information about workstations, execution initiation and execution procedures),
3. output formats,
4. returned error messages.

A reference list that the user may need is at the end of this user's guide.

A.1 Preparation of input

The input of GRPVSS can be classified into three types, one for world modelling files, another for user-supplied subroutines, and the third for path input. They are separately described in the following three sub-sections. The data for the IBM 7545 robot are listed as sample. The words with bold font are the required data in the files.

A.1.1 Input files for world modelling

For world modelling (a robot and its environment), the inputs include the following items:

1. the kinematic parameters of the to-be-simulated robot according to DH convention,
2. the maximum acceleration and deceleration of each joint of the robot,
3. the thickness of each link of the manipulator,
4. the coordinates of the vertices of the convex polyhedra which are used to model the manipulator,
5. the smallest angle of the convex polyhedra which are used to model the obstacles,
6. the coordinates of the vertices of the convex polyhedra which are used to model the robot's working environment (obstacles).

Item 1-4 and item 5-6 are separately stored in two files. They are individually described in the following two subsections.

A.1.1.1 Input files for robot modelling

Item 1-4 which model the robot are stored in a file with the file name, FILE, and user specified filetype (7 characters maximum). The format of all items of this robot-modelling file are listed in order in the file below.

1. kinematic parameters (item 1) and maximum joint accelerations and maximum joint decelerations (item 2):

N

N : total number of links of the robot

Sequentially list the following values for every joint (i = 1 - N).

IND_i

α_i, d_i, a_i, $\theta_{h,i}$, $\theta_{u,i}$, $\theta_{l,i}$, $acc_{m,i}$, $dec_{m,i}$(for revolute joints)

θ_i, α_i, a_i, $d_{h,i}$, $d_{u,i}$, $d_{l,i}$, $acc_{m,i}$, $dec_{m,i}$(for prismatic joints)

IND_i is the indicator of the type of joint i

0 : revolute joint

1 : prismatic joint

θ_i, a_i, α_i, d_i are the kinematic parameters of joint i

$\theta_{h,i}$, $\theta_{u,i}$, $\theta_{l,i}$ are the corresponding home position, upper limit, and lower limit of θ value of a revolute joint i.

$d_{h,i}$, $d_{u,i}$, $d_{l,i}$ are the corresponding home position, upper limit, and lower limit of d value of a prismatic joint i.

$acc_{m,i}$ is the maximum allowable joint acceleration of joint i

$dec_{m,i}$ is the maximum allowable joint deceleration of joint i

2. the thickness of each link (item 3) and coordinates of all convex polyhedra which form each link (item 4)

Sequentially list the following data for all links (0 - N).

D_i

D_i : the thickness of link i (Omit this value for link 0, manipulator base.)

$NOBAR_i$

$NOBAR_i$: number of hulls which form link i

Sequentially list the coordinates of all convex hulls (1 - $NOBAR_i$) of link i.

$x_{v,1}, y_{v,1}, z_{v,1}$

$x_{v,2}, y_{v,2}, z_{v,2}$

.
.
.

$x_{v,16}, y_{v,16}, z_{v,16}$

$x_{v,j}, y_{v,j}, z_{v,j}$ are coordinates of vertex i described w.r.t. the LCS.

It should be noted that the order of the vertices should follow the order shown on Figure 7 on page 55.

SAMPLE

```
4
0
0.0,25.5,15.8,0.0,0.0,200.0,2.,1.8
0
180.0,0.0,9.8,0.0,0.0,135.0,2.,1.8
1
0.0,0.0,0.0,0.0,0.0,9.8,0.7,0.6
0
0.0,15.05,0.0,0.0,-180.0,180.0,2.,1.8
4
5.875,2.125,0
0,2.5,0
-5.875,2.125,0
-6.25,-4.125,0
-5.875,-10.375,0
0,-10.5,0
5.875,-10.375,0
6.25,-4.125,0
5.875,2.125,2
0,2.5,2
-5.875,2.125,2
-6.25,-4.125,2
-5.875,-10.375,2
0,-10.5,2
5.875,-10.375,2
6.25,-4.125,2
2.5,-1.625,30.5
0.,-1.625,30.5
-2.5,-1.625,30.5
-2.5,-5.625,30.5
-2.5,-9.125,30.5
0.,-9.125,30.5
2.5,-9.125,30.5
2.5,-5.125,30.5
3.25,-1.625,2.0
0.,-1.625,2.0
-3.25,-1.625,2.0
-3.25,-5.625,2.0
-3.25,-9.125,2.0
0.,-9.125,2.0
3.25,-9.125,2.0
3.25,-5.125,2.0
3.5,1.625,28.5
0.,1.625,28.5
```

```
-3.5,1.625,28.5
-3.5,0.,28.5
-3.5,-1.625,28.5
0.,-1.625,28.5
3.5,-1.625,28.5
3.5,0.,28.5
3.5,1.625,41.5
0.,1.625,41.5
-3.5,1.625,41.5
-3.5,0.,41.5
-3.5,-1.625,41.5
0.,-1.625,41.5
3.5,-1.625,41.5
3.5,0.,41.5
2.5,6.375,22.5
0.,6.375,22.5
-2.5,6.375,22.5
-2.5,2.375,22.5
-2.5,-1.625,22.5
0.,-1.625,22.5
2.5,-1.625,22.5
2.5,2.375,22.5
2.5,6.375,13.5
0.,6.375,13.5
-2.5,6.375,13.5
-2.5,2.375,13.5
-2.5,-1.625,13.5
0.,-1.625,13.5
2.5,-1.625,13.5
2.5,2.375,13.5
3.23
2
-17.0,1.625,0.
-17.0,1.625,-3
-17.0,0.,-3
-17.0,-1.625,-3
-17.0,-1.625,0
-17.0,-1.625,3
-17.0,0.,3
-17.0,1.625,3
1.0,1.625,0.
1.0,1.625,-3
1.0,0.,-3
1.0,-1.625,-3
1.0,-1.625,0.
1.0,-1.625,3
1.0,0.,3
1.0,1.625,3
1.0,1.615,3
1.0,0.,3
1.0,-1.615,3
-1.25,-1.615,3
-3.5,-1.615,3
-3.5,0.,3
-3.5,1.615,3
-1.25,1.615,3
1,1.615,14
1,0.,14
1,-1.615,14
-1.25,-1.615,14
-3.5,-1.615,14
-3.5,0.,14
-3.5,1.615,14
-1.25,1.615,14
4.
2
1.0,-1.5,-3.
1.0,-1.5,0.
1.0,-1.5,3.
1.0,0.,3.
1.0,1.5,3
1.0,1.5,0.
1.0,1.5,-3
1.0,0.,-3
-11.0,-1.5,-3
-11.0,-1.5,0.
-11.0,-1.5,3
-11.0,0.,3
-11.0,1.5,3
-11.0,1.5,0.
-11.0,1.5,-3
-11.0,0.,-3
1.,-1.5,-3
-0.75,-1.5,-3
-2.5,-1.5,-3
-2.5,0.,-3
-2.5,1.5,-3
-0.75,1.5,-3
1.,1.5,-3
1.,0,-3
1.,-1.5,-15
-0.75,-1.5,-15
-2.5,-1.5,-15
-2.5,0.,-15
-2.5,1.5,-15
-0.75,1.5,-15
1.,1.5,-15
1.,0.,-15
1.0
2
0.5,-0.5,-10.0
0.,-0.5,-10.0
-0.5,-0.5,-10.
-0.5,0.,-10.
-0.5,0.5,-10.
0.,0.5,-10.
0.5,0.5,-10.
0.5,0.,-10.
0.5,-0.5,7.
0.,-0.5,7.
-0.5,-0.5,7.
-0.5,0.,7.
-0.5,0.5,7.
0.,0.5,7.
0.5,0.5,7.
0.5,0.,7.
1.,-1.,7
0.,-1.,7
-1.,-1.,7
-1.,0.,7
-1.,1.,7
0.,1.,7
1.,1.,7
1.,0.,7
1.,-1.,9
0.,-1.,9
-1.,-1.,9
-1.,0.,9
-1.,1.,9
0.,1.,9
1.,1.,9
1.,0.,9
1.5
1
0.35,-0.35,0
0,-0.5,0
-0.35,-0.35,0
-0.5,0,0
-0.35,0.35,0
0,0.5,0
0.35,0.35,0
0.5,0,0
0.35,-0.35,-7.0
0,-0.5,-7.
-0.35,-0.35,-7.
-0.5,0,-7.
-0.35,0.35,-7.
0,0.5,-7.
0.35,0.35,-7.
0.5,0,-7.
```

A.1.1.2 Input file for environment modelling

Item 5-6 model the robot's working environment. They are stored in a file with the file name, FILE, and user specified filetype(7 characters maximum). The format of all items of this environment modelling file are listed in order in the file below.

1. The smallest angle of all convex hulls which model the working environment.

   θ_s

2. The smallest thickness of all convex hulls which model the working environment.

   L

3. The coordinates of all convex hulls which model the working environment.

   NOB

   NOB : total number of convex hulls which model the working environment.

   Sequentially list the coordinates of all convex hulls (1 - NOB).

   $x_{v,1}, y_{v,1}, z_{v,1}$

   $x_{v,2}, y_{v,2}, z_{v,2}$

   .
   .
   .

   $x_{v,16}, y_{v,16}, z_{v,16}$

   $x_{v,j}, y_{v,j}, z_{v,j}$ are coordinates of vertex i described w.r.t. WCS.

   It should be noted that the order of the vertices should follow the order shown on Figure 7 on page 55.

SAMPLE

```
90.
5.
2
15,15,15
17.5,14,15
20,15,15
21,17.5,15
20,20,15
17.5,21.,15
15,20,15
14.,17.5,15
15,15,20
17.5,14,20
20,15,20
21,17.5,20
20,20,20
17.5,21.,20
15,20,20
14.,17.5,20
-12,10,15
-14.5,9,15
-17,10,15
-18,12.5,15
-17,15,15
-14.5,16.,15
-12,15,15
-11.,12.5,15
-12,10,20
-14.5,9,20
-17,10,20
-18,12.5,20
-17,15,20
-14.5,16.,20
-12,15,20
-11.,12.5,20
```

A.1.2 Specifications of user-supplied subroutines

Two subroutines need to be supplied by the user. They are:

1. a complied subroutine, named INVERS, which is used to solve the inverse kinematics problems of the to-be-simulated robot, (Subroutine INVERS is necessary when CP operations or PTP operations described w.r.t. WCS are applied. This subroutine for IBM 7545 robot is already installed in the system.)

2. a complied subroutine, named MOTFIL, which includes the motion-time profiles of the to-be-simulated robot (optional).

The required specifications of these two subroutines are described in the following.

Subroutine INVERS receives position and orientation of the end effector from GRPVSS and returns the joint values of the manipulator at that particular configuration.

SUBROUTINE INVERS(X, Y, Z, OR, PAR, IERROR)

X, Y, Z : coordinates of the position of the end effector
short floating numbers (from GRPVSS)

OR(3,3) : the orientation matrix of the end effector
short floating numbers (from GRPVSS)

PAR(N) : joint value of the manipulator when the end effector is at the pre-defined state.
short floating numbers (returned by INVERS)
N : total number of links

IERROR : error indicator,
0 : No error is detected.
1 : The defined state is out of the workspace.
fullword integer (returned by INVERS)

Subroutine MOTFIL is required to return the joint values and joint velocities of the manipulator when a particular motion time measured from the starting point is given by GRPVSS.

SUBROUTINE MOTFIL(i, $a_{m,i}$, $d_{m,i}$, $VELDES_i$, $START_i$, END_i, T, VEL_i, POS_i, INDEND, $INDS_i$)

i : the joint number

fullword integer (from GRPVSS)

$a_{m,i}$: the maximum allowable acceleration of joint i

short floating number(from GRPVSS)

$d_{m,i}$: the maximum allowable deceleration of joint i

short floating number(from GRPVSS)

$VELDES_i$: the desired joint velocity of joint i

short floating number(from GRPVSS)

$START_i$: the joint value of joint i at starting position

short floating number(from GRPVSS)

END_i : the joint value of joint i at ending position

short floating number(from GRPVSS)

T : the motion time value measured from the beginning.

short floating number(from GRPVSS)

VEL_i : the joint i velocity at time T

short floating number(returned by MOTFIL)

POS_i : the joint i position at time T

short floating number(returned by MOTFIL)

INDEND : ending indicator

0 : joint i does not reach the END_i at time T.

1 : joint i reaches the END_i at time T.

full word integer(returned by MOTFIL)

$INDS_i$: Assign an arbitrary integer variable here.

full word integer(from GRPVSS)

Even if these two subroutines are not needed in the applications, it should be noted that these two subroutines should still be prepared with only RETURN and END statements. Otherwise, the warning message of undefined names will be found when GRPVSS is executed.

A.1.3 Input files for path input

The to-be-simulated path may be input by user on the screen or through an input file. Here the format of the path-input file is described.

NOMS

NOMS : total number of motion segments in this path.

Sequentially list the specifications of all motion segments (1 - NOMS).

IND

IND : indicator of the operation type of this motion segment

1 : PTP (described w.r.t. JCS)

2 : PTP (described w.r.t. WCS)

3 : CP

The specifications for the corresponding operation type of this segment

For PTP(JCS)

$q_{f,1}$, **VELDES**$_1$

$q_{f,2}$, **VELDES**$_2$

.
.
.

$q_{f,N}$, **VELDES**$_N$.

$q_{f,i}$: the joint value at destination,

q is θ for revolute joint(measured in degrees).

q is d for prismatic joint.

$VELDES_i$: the desired joint velocity of this motion segment

(measured in degrees/sec for revolute joints)

For PTP(WCS)

XEND, YEND, ZEND, KX, KY, KZ, KTHETA

XEND, *YEND*, *ZEND* : position of the end effector at destination

KX, *KY*, *KZ* : the axis about which the orientation of the end effector changes

KTHETA : the rotating angle about the previously defined axis

VELDES$_1$

VELDES$_2$

.
.

$VELDES_N$

For CP

XEND, YEND, ZEND, KX, KY, KZ, KTHETA

SAMPLE

```
3
1
50.,3.
50.,3.
5.,0.3
50.,3.
2
0.,20.,8.,0.,0.,-1.,30.
3.
3.
0.3
3.
3
-20.,-5.,2.,0.,0.,-1.,30.
```

A.2 Running procedures

GRPVSS currently can only be run on Virginia Tech's VTVM3 mainframe computer system because the employed graphics package, graPHIGS, is installed on VTVM3. The memory space for GRPVSS program and associated user-supplied subroutines, as well as the data files needs approximately 450 blocks of disk storage. Therefore, a VTVM3 computer account with at least 500 blocks memory space is needed. Some device-dependent graphics specifications used in the current version of GRPVSS are only fit for the IBM 5080 graphics workstation. However, they can be changed for other graphics workstations. Beside the IBM 5080 graphics workstation, the workable workstations include the IBM 3250 graphics workstation, IBM GDDM graphics workstations and GDF graphics workstation. The specifications for certain workstations can be checked from the

Writing Application with graPHIGS in the reference list which is at the end of this user's guide. Here the running procedures are described based on the IBM 5080 graphics workstation.

A.2.1 Initiation

Before running GRPVSS, the user is recommended to use the checklist shown in Table 4 to rapidly determine if every necessary input file or subroutine is available. Use the jump screen on the IBM 5080 graphics workstation to log on. The jump screen is reached by pressing Alt and PA3 keys concurrently. The log-on procedure is the standard procedure used for the Virginia Tech mainframe computer system. After logon, a graphics workstation needs to be attached to the user's computer account. Press the Alt and PA3 keys concurrently to go out of the jump screen. Press the Reset and Enter keys concurrently to activate the Graphics Task Monitor(GTM). Wait for the GTM screen to appear, type USER-ID on the first line which requests user identification, use the JUMP key to go to the third line which asks "IS USERID TO BE AUTOLOGGED (YES/NO)?", type NO on this line, and hit the Enter key. Go back to the jump screen again. A few seconds later, a system returned message, "GRAF 200 ATTACHED", should appear which means the graphics workstation is attached to your account. Now you are ready to run GRPVSS as long as everything in the checklist has been done. Simply type GRPVSS and hit the Enter key to execute GRPVSS. When the "EXECUTION BEGIN..." message appears, press the Alt and PA3 keys concurrently to go to the graphics screen. After completing all above procedures, GRPVSS will give instructions to the user at the menu area of the screen whenever GRPVSS needs input or awaits user options. It should be noted that the operations can be broken by pressing the Alt and Cncl keys concurrently in most input modes and the operation returns to the main menu. In the following section, a running sample is demonstrated to explain the running procedures.

Table 4. Checklist for GRPVSS

INPUTS	READY ?
ROBOT MODELLING FILE	
ENVIRONMENT MODELLING FILE	
SUBROUTINE MOTFIL (for motion time profiles)	
SUBROUTINE INVERS (for inverse kinematics problems)	
PATH INPUT FILE (optional)	

A.2.2 Running Sample

The running sample is simulating the IBM 7545 robot in a created environment formed by 7 convex polyhedra. The bold font words show the message in the screen. The bold italic font words show the data to be entered.

WHICH ROBOT ARE YOU GOING TO SIMULATE ?

1. IBM 7545 ROBOT
2. OTHERS

ENTER ONE OF THE OPTION NUMBERS

1

The option is chosen for running the IBM 7545 robot. If another type of robot is chosen, the robot modelling parameters need to be entered as the procedures in the following box.

THE KINEMATIC AND SHAPE PARAMETERS OF THE

TO-BE-SIMULATED ROBOT SHOULD BE IN THE FILE WITH

FILENAME: FILE FILETYPE: USER SPECIFIED

ENTER THE FILETYPE (7 CHARACTERS MAXIMUM)

ROB

A file (FILE ROB) is chosen as the data file to model the robot. The file format is described in Section A.1.1.1.

ENTER THE WAY TO GENERATE THE PTP TRAJECTORY

1. IDEAL TRAPEZOID FUNCTIONS
2. MOTION-TIME PROFILES(FOR THIS OPTION, YOU MUST ALREADY HAVE A SUBROUTINE IN A COMPLIED FILE NAMED "MOTFIL". MAKE SURE IT IS AVAILABLE.

1

Ideal trapezoid functions are chosen in this case. If the motion-time profile option is chosen, the complied subroutine "MOTFIL" should be available as indicated on the screen.

THE VERTICES OF THE OBSTACLES IN THE WORKING

ENVIRONMENT SHOULD BE IN THE FILE WITH

FILENAME: FILE FILETYPE: USER SPECIFIED

ENTER THE FILETYPE (7 CHARACTERS MAXIMUM)

OBSTAC

A file (FILE OBSTAC) is chosen as the data file to model the robot's working environment. The file format is described in Section A.1.1.2.

ENTER THE TOLERANCE VALUE

0.5

The tolerance value is set to be 0.5 at this time. Units should be consistent with the one used in robot modelling.

After these inputs, the image of the robot and its environment will be displayed on the screen. The data including the joint values and position and orientation of the end effector at the current configuration are shown in the left part of the screen.

WANT TO CHANGE THE VIEW ANGLES ?

ENTER Y/N (DEFAULT = N)

y

This procedure allows you to change the view angles to the best one. If the view angles are appropriate, hit ENTER key to skip the next step.

ENTER THE DESIRED VIEW ANGLES IN DEGREE

ENTER ANGLEX, ANGLEY, ANGLEZ,

USE A COMMA AFTER EACH DATA INCLUDING THE LAST DATA

-75.,0.,85.,

The default view angles are -75.,0.,170.. The rotating procedures are described below. Everytime you change view angles, you can assume that the X axis is pointing to the right of the screen and the Z axis is pointing outward from the screen. Use this coordinate frame to determine the view angles. The coordinate frame rotates ANGLEX degrees about the current X axis first. Then the coordinates frame rotates ANGLEY degrees about the updated Y axis. Finally, the coordinates frame rotates ANGLEZ degrees about the updated Z axis.

WANT TO CHANGE THE SCALE AND SHIFT THE WINDOW ?

ENTER Y/N (DEFAULT = N)

y

This procedure allows you to change the scale factor and shift the viewing window to make the screen exactly fit the working environment or do the zooming for your convenience. If the screen image is fine, hit ENTER key to skip the next step.

ENTER THE SCALING FACTOR (SCALE)

AND THE SHIFTING OF WINDOW (SHIFTX, SHIFTY, SHIFTZ)

ENTER SCALE, SHIFTX, SHIFTY, SHIFTZ,

USE A COMMA AFTER EACH DATA INCLUDING THE LAST ONE

1.5,-20.,-10.,0.,

The default scale factor is 1.0. The default shifting values are 0.,0.,0.. Everytime you change the scale factor, you factor the original dimension. Shifting is also performed before scaling; therefore, shifting values should be determined by the original dimensions. The original view scope is -50. to 50. in each dimension (X, Y, Z dimension).

ENTER THE WAY YOU WANT TO INPUT THE PATH

1. FROM DATA FILE

2. INTERACTIVELY INPUT ON THE SCREEN

1

Here the data is chosen to be the input from the file.

THE SPECIFICATIONS OF ALL SEGMENTS OF THE PATH SHOULD

BE STORED IN THE FILE WITH THE FOLLOWING NAME

FILENAME: FILE FILETYPE: USER SPECIFIED

ENTER THE FILETYPE (7 CHARACTERS MAXIMUM)

PATH

A file (FILE PATH) is chosen as the data file. The format of this file can be referenced in Section A.1.3.

DO YOU WISH TO STORE THE RESULTS OF THIS PATH

IN A FILE ?

ENTER Y/N (DEFAULT = Y)

Y

The results need to be stored in a file. If "N" is entered, the next step is skipped and the processing is initiated.

THE RESULTS WILL BE STORED IN A FILE NAMED

FILENAME: FILE FILETYPE: USER SPECIFIED

ENTER THE FILETYPE (7 CHARACTERS MAXIMUM)

MAKE SURE THE FILETYPE YOU SPECIFIED DOES NOT EXIST

IN THE DISK.

OUT1

The output file is named as FILE OUT1. The result will be stored in this file. If one file with this name already exists in the disk, the execution of GRPVSS will be terminated.

PROCESSING .....

GRPVSS is processing the path.

WANT TO CHANGE THE VIEW ANGLES ?

ENTER Y/N (DEFAULT = N)

N

The assigned path has been processed. Now the program will do the graphics animation on the screen. Whether there is collision in this path or not will declared in the left column of the screen. If there is any error in the data file, the error message will be returned on the screen and on the data file. Error messages will be listed in a later section.

WANT TO CHANGE THE SCALE AND SHIFT THE WINDOW ?

ENTER Y/N (DEFAULT = N)

N

WHAT TYPE OF MOTION IMAGE DO YOU LIKE?

CONTINUOUS MOTION OR STEPPING MOTION

IF STEPPING, ENTER 1 AND HIT ENTER KEY FOR EACH STEP

IF CONTINUOUS, ENTER ANY CHARACTER OTHER THAN 1

2

A continuous motion image is chosen. Images of the manipulator at all simulation steps are displayed step by step continuously. At each step, the detected interference points are shown on the screen with red star symbols. The total interference point number at each step is also shown in the left column. If stepping motion is chosen, the image of the next step will be displayed until the user hits the Enter key.

ENTER ONE OF THE FOLLOWING OPTION NUMBERS?

1. SIMULATE ANOTHER ROBOT

2. CHANGE THE ENVIRONMENT

3. CHANGE THE TOLERANCE VALUE

4. PERFORM ANOTHER MOTION

5. SEE THE ANIMATION AGAIN

6. EXIT

4

After the animation is finished, the program will come to this main menu. In this case, 4 is specified. That is, another path will be simulated in the same environment with the same tolerance value.

ENTER THE WAY YOU WANT TO INPUT THE PATH

1. FROM DATA FILE

2. INTERACTIVELY INPUT ON THE SCREEN

2

Here the path is entered on the screen.

WHICH TYPE OF OPERATION FOR THIS MOTION SEGMENT ?

1. PTP OPERATION SPECIFIED IN JOINT COORD. SYSTEM

2. PTP OPERATION SPECIFIED IN WORLD COORD. SYSTEM

3. CP STRAIGHT-LINE OPERATION

1

The PTP operation specified in JCS is assigned for this path. GRPVSS will ask for the specifications of this operation.

ENTER THE JOINT VALUE OF JOINT 1 AT ENDING POINT

MIN VALUE = 0.00 MAX VALUE = 200.00 HOME POSITION = 0.00

AND THE DESIRED JOINT VELOCITY

USE A COMMA AFTER EACH DATA INCLUDING THE LAST ONE

50.,4.,

Revolute joints are measured in degrees. Here, joint 1 must reach to 50°, and the desired joint velocity is 4° per second. If an unreachable joint value is entered, an error message will be displayed that asks for the values again. This input procedure will be iterated until all final joint values and desired joint velocities are entered. Here, for convenience, the values of the remaining joints have been omitted. Break action (pressing Alt and Cncl keys concurrently) can let the program go to the main menu.

.
.
.

ARE THE INPUT DATA CORRECT ?

ENTER Y/N (DEFAULT = Y)

Y

If "n" is entered, the specifications for this operation will be requested again.

DO YOU WISH TO STORE THE RESULTS OF THIS PATH

IN A FILE ?

ENTER Y/N (DEFAULT = Y)

n

The results will not be stored in a file.

PROCESSING .....

GRPVSS is processing this path.

WANT TO CHANGE THE VIEW ANGLES ?

ENTER Y/N (DEFAULT = N)

N

The assigned path has been processed. Now, the program will do the graphics animation on the screen. If there is any error in this assigned path, the error message is displayed on the screen. The procedures of graphics animation is same as described before. Therefore, they are omitted and the demonstration goes to the main menu directly.

.
.
.

ENTER ONE OF THE FOLLOWING OPTION NUMBERS?

1. SIMULATE ANOTHER ROBOT

2. CHANGE THE ENVIRONMENT

3. CHANGE THE TOLERANCE VALUE

4. PERFORM ANOTHER MOTION

5. SEE THE ANIMATION AGAIN

6. EXIT

4

Another path will be simulated in the same environment with the same tolerance value.

ENTER THE WAY YOU WANT TO INPUT THE PATH

1. FROM DATA FILE

2. INTERACTIVELY INPUT ON THE SCREEN

2

The path is entered on the screen.

WHICH TYPE OF OPERATION FOR THIS MOTION SEGMENT ?

1. PTP OPERATION SPECIFIED IN JOINT COORD. SYSTEM

2. PTP OPERATION SPECIFIED IN WORLD COORD. SYSTEM

3. CP STRAIGHT-LINE OPERATION

2

The PTP operation specified w.r.t. WCS is assigned for this path. GRPVSS will ask for the specifications of this operation.

ENTER THE FINAL POSITION (X, Y, Z) OF THE END EFFECTOR

USE A COMMA AFTER EACH DATA INCLUDING THE LAST ONE

0.0,20.0,6.0,

The desired position of the end effector is entered.

ENTER THE AXIS (KX,KY,KZ) ABOUT WHICH THE

ORIENTATION OF END EFFECTOR CHANGES

AND THE CORRESPONDING CHANGING ANGLE (KTHETA) IN DEGREES

USE A COMMA AFTER EACH DATA INCLUDING THE LAST ONE

0.0,0.0,-1.,30.,

The orientation of the end effector at the final position is found by entering the changing axis and the changing angle. GRPVSS will calculate the orientation matrix from these data. In this case, subroutine IBMINV which solves inverse kinematics problems for the IBM 7545 robot is employed to solve the joint values at the ending point. If another robot is simulated, the user-supplied INVERS subroutine is used to solve the inverse kinematics problems. When the specified ending robot state is out of the workspace, an error message will be displayed on the screen and the specifications for this operation will be requested again. Break action lets the program go to the main menu.

ENTER THE DESIRED JOINT VELOCITY OF JOINT 1

FOR REVOLUTE JOINTS, THE UNIT IS DEGREE/SEC

USE A COMMA AFTER EACH DATA INCLUDING THE LAST ONE

4.0,

The desired joint velocity is 4° per second for joint 1. These input procedures will iterate until the desired joint velocities of all joints are entered. (Here the procedures for entering values of the remaining joints are omitted.) Break action can let the program go to the main menu.

.
.
.

ARE THE INPUT DATA CORRECT ?

ENTER Y/N (DEFAULT = Y)

Y

The same procedures that request filing the result, processing the path, and graphics animating as described above will be performed. Those are omitted here. The main-menu is directly reached again.

.
.
.

ENTER ONE OF THE FOLLOWING OPTION NUMBERS

1. SIMULATE ANOTHER ROBOT

2. CHANGE THE ENVIRONMENT

3. CHANGE THE TOLERANCE VALUE

4. PERFORM ANOTHER MOTION

5. SEE THE ANIMATION AGAIN

6. EXIT

4

Another path will be simulated in the same environment with the same tolerance value.

ENTER THE WAY YOU WANT TO INPUT THE PATH

1. FROM DATA FILE

2. INTERACTIVELY INPUT ON THE SCREEN

2

Here the path is entered on the screen.

WHICH TYPE OF OPERATION FOR THIS MOTION SEGMENT ?

1. PTP OPERATION SPECIFIED IN JOINT COORD. SYSTEM

2. PTP OPERATION SPECIFIED IN WORLD COORD. SYSTEM

3. CP STRAIGHT-LINE OPERATION

3

CP straight-line operation is assigned for this path. GRPVSS will ask for the specifications of this operation.

ENTER THE FINAL POSITION (X, Y, Z) OF THE END EFFECTOR ?

USE A COMMA AFTER EACH DATA INCLUDING THE LAST ONE

0.0,20.0,6.0,

The desired position of the end effector is entered.

ENTER THE AXIS (KX,KY,KZ) ABOUT WHICH THE

ORIENTATION OF THE END EFFECTOR CHANGES

AND THE CORRESPONDING CHANGING ANGLE (KTHETA) IN DEGREES

USE A COMMA AFTER EACH DATA INCLUDING THE LAST ONE

0.0,0.0,-1.,30.,

GRPVSS calculates the joint values of the manipulator at the final position. If the specified ending robot state is out of workspace, an error message will be shown on the screen and the specifications for this operation will be requested again. Break action lets the program go to the main menu.

ARE THE INPUT DATA CORRECT ?

ENTER Y/N (DEFAULT = Y)

Y

This confirms that the inputs are correct. The same procedures of asking for filing the result, processing the path, and graphics animating will be performed. Those are also omitted. The main-menu is directly reached again.

.
.
.

ENTER ONE OF THE FOLLOWING OPTION NUMBERS.

1. SIMULATE ANOTHER ROBOT

2. CHANGE THE ENVIRONMENT

3. CHANGE THE TOLERANCE VALUE

4. PERFORM ANOTHER MOTION

5. SEE THE ANIMATION AGAIN

6. EXIT

6

Option 6 terminates the simulation. For other options, the program will go to the corresponding stage and again perform the procedures as described above. For example, if changing the tolerance value option is chosen, the program will go to the step of changing the tolerance value as we described previously and perform in the same procedures until it returns to the main menu.

GRPVSS is designed to be menu-driven. Once the user prepares all the necessary input files and subroutines, the user will have no problem in the simulation procedures because very clear instructions are shown on the screen at every stage.

A.3 Output format

As described above, the results of the simulation can be stored in an output file. The information includes the tolerance value, the operation type and corresponding specifications of each motion segment, the joint values of the manipulator at each simulation step, the interference points, and the error message. The descriptions about the data in the output file are very clear. A sample is listed in the following.

OUTPUT FILE SAMPLE

```
THE TOLERANCE VALUE =   1.00

-------------------------------------------------------------------------
THE  1TH SEGMENT OF THE PATH INPUT FROM FILE PATH
-------------------------------------------------------------------------

          --- PTP (WCS)---
 ENDING POSITION    ORIENTATION CHANGE AXIS    ROTATING ANGLE

 0.00  20.00   8.00              0.00   0.00  -1.00              30.00

 JOINT VELOCITY     3.00    3.00    0.30    3.00

 IN    1TH STEP, THE JOINT VALUES ARE
       1.05      1.05      0.24      1.05

 THERE ARE    0 INTERFERENCE POINTS IN THIS STEP


 IN    2TH STEP, THE JOINT VALUES ARE
       5.26      5.26      0.69      5.26

 THERE ARE    0 INTERFERENCE POINTS IN THIS STEP


 IN    3TH STEP, THE JOINT VALUES ARE
       8.25      8.25      0.99      8.25

 THERE ARE    0 INTERFERENCE POINTS IN THIS STEP


 IN    4TH STEP, THE JOINT VALUES ARE
      11.20     11.20      1.28     11.20

 THERE ARE    0 INTERFERENCE POINTS IN THIS STEP
```

.
.
.
.

IN 33TH STEP, THE JOINT VALUES ARE
61.16 79.90 2.45 163.95

THERE ARE 0 INTERFERENCE POINTS IN THIS STEP

IN 34TH STEP, THE JOINT VALUES ARE
61.16 79.90 2.45 171.06

THERE ARE 0 INTERFERENCE POINTS IN THIS STEP

THE 2TH SEGMENT OF THE PATH INPUT FROM FILE PATH

--- CP ---
ENDING POSITION ORIENTATION CHANGE AXIS ROTATING ANGLE

-20.00 -5.00 2.00 0.00 0.00 -1.00 30.00

IN 35TH STEP, THE JOINT VALUES ARE
62.69 95.39 3.00 -169.16

THERE ARE 0 INTERFERENCE POINTS IN THIS STEP

IN 36TH STEP, THE JOINT VALUES ARE
67.36 107.88 3.56 -149.23

THERE ARE 0 INTERFERENCE POINTS IN THIS STEP

.
.
.
.
.

IN 40TH STEP, THE JOINT VALUES ARE
112.45 126.75 5.77 -74.22

THERE ARE 0 INTERFERENCE POINTS IN THIS STEP

IN 41TH STEP, THE JOINT VALUES ARE
125.16 122.05 6.32 -63.45

THERE ARE 8 INTERFERENCE POINTS IN THIS STEP

-11.40	4.00	9.00
-11.40	4.00	3.13
-14.46	4.00	3.13
-14.46	4.00	9.00
-11.33	3.76	9.00
-14.10	2.85	9.00
-13.01	2.31	9.00
-11.86	2.68	9.00

IN 42TH STEP, THE JOINT VALUES ARE
136.72 114.01 6.87 -57.16

THERE ARE 21 INTERFERENCE POINTS IN THIS STEP

-13.17	0.00	9.00
-15.60	4.00	8.63
-16.01	4.00	8.63
-15.60	4.00	9.00
-16.01	4.00	9.00
-16.28	0.00	9.00
-13.17	0.00	8.63
-16.28	0.00	8.63
-12.85	0.92	9.00
-12.19	2.81	9.00
-14.08	3.47	9.00
-16.62	2.24	9.00
-17.28	0.35	9.00
-13.18	1.39	9.00
-13.48	2.56	9.00
-14.54	3.14	9.00
-15.71	2.83	9.00
-16.30	1.77	9.00
-15.99	0.60	9.00
-14.93	0.02	9.00
-13.76	0.33	9.00

IN 43TH STEP, THE JOINT VALUES ARE
147.39 103.13 7.42 -54.62

THERE ARE 14 INTERFERENCE POINTS IN THIS STEP

-14.20	0.00	9.00
-18.44	0.00	9.00
-14.20	0.00	8.08
-18.44	0.00	8.08
-14.03	0.50	9.00
-15.91	1.16	9.00
-17.80	1.83	9.00
-15.24	0.00	9.00
-17.85	0.00	9.00
-15.24	0.00	2.03
-17.85	0.00	2.03
-15.28	0.19	9.00
-16.31	0.83	9.00
-17.49	0.58	9.00

IN 44TH STEP, THE JOINT VALUES ARE
157.66 89.49 7.98 -55.22

THERE ARE 0 INTERFERENCE POINTS IN THIS STEP

IN 45TH STEP, THE JOINT VALUES ARE
166.68 75.16 8.45 -58.16

THERE ARE 9 INTERFERENCE POINTS IN THIS STEP

-18.53	-5.00	1.00
-18.53	-5.00	2.00
-21.47	-5.00	1.00
-21.47	-5.00	2.00
-20.00	-5.00	1.00

```
-18.48      -5.41      2.00
-21.38      -5.79      2.00
-20.41      -6.52      2.00
-19.21      -6.38      2.00

-----------------------------------------------------------------------
THE  3TH SEGMENT OF THE PATH INPUT FROM FILE PATH
-----------------------------------------------------------------------


        --- PTP (JCS)---
            STARTING JOINT VALUE   FINAL JOINT VALUE     DESIRED VELOCITY

JOINT  1              166.68              50.00               3.00
JOINT  2               75.16              50.00               3.00
JOINT  3                8.45               5.00               0.30
JOINT  4              -58.16              50.00               3.00

IN  46TH STEP, THE JOINT VALUES ARE
   165.16    73.64     8.14    -56.64

THERE ARE    8 INTERFERENCE POINTS IN THIS STEP

   -18.99      -5.00      1.31
   -21.78      -5.00      1.31
   -21.78      -5.00      2.00
   -18.99      -5.00      2.00
   -20.00      -5.00      1.31
   -21.78      -5.02      2.00
   -20.88      -5.82      2.00
   -19.67      -5.77      2.00

IN  47TH STEP, THE JOINT VALUES ARE
   160.84    69.32     7.71    -52.32

THERE ARE    5 INTERFERENCE POINTS IN THIS STEP

   -20.88       0.00      9.00
   -21.86       0.00      9.00
   -20.88       0.00      7.79
   -21.86       0.00      7.79
   -21.46       0.48      9.00
                  .
                  .
                  .
                  .
                  .

IN  82TH STEP, THE JOINT VALUES ARE
    52.34    50.00     5.00    50.00

THERE ARE    0 INTERFERENCE POINTS IN THIS STEP


IN  83TH STEP, THE JOINT VALUES ARE
    50.00    50.00     5.00    50.00

THERE ARE    0 INTERFERENCE POINTS IN THIS STEP


-----------------------------------------------------------------------
THE  4TH SEGMENT OF THE PATH INPUT FROM FILE PATH
-----------------------------------------------------------------------
```

```
      --- CP ---
ENDING POSITION    ORIENTATION CHANGE AXIS      ROTATING ANGLE

15.00   2.00   8.00          0.00   0.00  -1.00                30.00

THIS PATH IS NOT QUALIFIED.
THE ERROR IS TYPE 3
CHECK THE USER GUIDE FOR DETAILED ERROR DESCRIPTIONS

------------------------------------------------------------------------
TOTAL   83 STEPS IN THIS PATH

  COLLISIONS DETECTED IN THIS PATH
```

A.4 Error messages

The error messages are described in the order of their error numbers in the following list.

Error 1:

The movement of the end effector in the CP operation is zero. In CP operations, the simulation resolution is determined by the increment of the end effector. Therefore, the simulation can not be performed if movement of the CP straight-line operation is zero. PTP operation is suggested if this error is not caused by a mistake.

Error 2:

Unreachable intermediate point was encountered. Simulation terminated at that point. This path is out of the workspace.

Error 3:

The final position of the motion segment I in the path input file is out of workspace. This motion segment was not simulated. The simulation terminated when this motion segment was encountered.

**The number I will be returned along with the error message for Error 3 - Error 6.

Error 4:

The movement of the end effector in the CP operation of motion segment I in the input path file is zero. PTP operation is suggested if the error is not caused by a mistake.

Error 5:

Unreachable intermediate point was encountered in the motion segment I in the input file. Simulation terminated at that point. This motion segment is out of the workspace.

Error 6:

Undefined operation type data was encountered in motion segment I in the input file. Simulation terminated at that point. Check the path input file.

References

- Programming guide for VS Fortran Version 2, Release 1.0, IBM SC26-4222-0.
- Understanding graPHIGS, Release 2.0, IBM SC33-8102-0.
- Writing Applications with graPHIGS, Release 2.0, IBM SC33-8103-1.
- Programmer's Reference for graPHIGS, Release 2.0, IBM SC33-8104-1.

The vita has been removed from the scanned document