 Journal of Applied Mechanisms & Robotics, Vol. 2, No. 2, April, 1995, pp. 20-33.

REAL-TIME FOUR-DIMENSIONAL COLLISION DETECTION FOR AN INDUSTRIAL ROBOT MANIPULATOR

Harry H. Cheng
Integration Engineering Laboratory
Department of Mechanical and Aeronautical Engineering
University of California
Davis, California 95616

ABSTRACT

A real-time collision detection technique for four-dimensional (three spatial dimensions and time) objects is presented in this paper. The developed collision detection method separates the geometrical information from time and complicated motion characteristics through s-formulations. The collision detection of moving objects then becomes a geometrical problem so that it can be solved in real time by using a modified bisection method. The algorithm can predict the exact time and location of the collision of moving objects even for cases when dimensions of moving objects change during the motion. Our collision detection algorithm has been integrated into the real-time control system of the UPSarm which is a ten degree-of-freedom hybrid serial-and-parallel-driven redundant robot manipulator. The capabilities and features of this collision detection technique are demonstrated by manipulating the UPSarm in an environment where an object presents. For our practical application, formulations of intersection for two rectangular solids and polyhedra based upon the s-formulation are also derived in detail in this paper.

INTRODUCTION

Currently, the task-level programming of robot manipulators is an active research topic. It has been recognized that in a task-level robot programming environment, the user will be able to program the robot in a significantly high level (Lozano-Perez, 1981). The system will have the ability to perform many tasks automatically. In such an intelligent system, a collision detection or collision avoidance capability is essential since a collision-free path for a robot manipulator or mobile robot must be planned by the robot itself.

Many collision avoidance approaches including the artificial potential function method (Khatib, 1986; Borenstein and Koren, 1989) and harmonic potential function method (Kim and Khosla, 1992) have been investigated. The direct sensing based collision avoidance system for slow moving objects has been reported by Cheung and Lumelsky (1990). Solving the findpath problem by the configuration space approach has been popularized by Lozano-Perez (1981, 1983) and his colleagues (Brook, 1983; Donald, 1987; Latombe, 1991). Most of the aforementioned research concern the point-to-point collision avoidance. In these systems, the goal is to move a robot from a known initial position to a final position free of collision, but, the trajectory of the robot is not concerned. Another interesting task which requires that the robot arm avoid collision with surrounding obstacles while the tool center point (TCP) of the end-effector of a robot manipulator follows a specified trajectory has been addressed in (Oh et al, 1984; Maciejewski and Klein, 1985).

Different from the above collision avoidance approaches, this paper addresses a more fundamental problem, collision detection of moving objects, which has many practical applications. A number of collision detection algorithms for moving objects has been developed in the past decade. Cameron (1985) classified the collision detection of moving objects into three categories: multiple interference detection, swept volume interference detection, and intersection detection. In the context of this paper, the interference refers to the state where two objects occupy the same area in the working space. The condition in which two objects just contact each other is a special case of the interference, which is referred to as the intersection of two objects. Both interference and intersection of objects are referred to as the collision of objects.

The multiple interference detection is intuitively simple. In this method, the positions and orientations of moving objects are sampled at a finite number of times and interference detection between objects is performed at each time step. For example, Roach and Boaz (1987) used this technique along with the region octree representation of object and workspace to detect the collision of multiple robot arms working in a common workspace. For a real-time efficiency, the collision avoidance algorithm developed by Shaffer and Herb (1992) uses N-object octrees instead of region octrees to model the workspace. There are some problems associated with this method in the real-time implementation. If the sampling time step is too large, a possible collision might be missed. On the other hand, if the step is too small, the computation time might be too long to be completed in real time. In a typical real-time implementation, each interference detection is performed at the trajectory update rate. However, this implementation cannot accommodate the changes of a dynamically varying environment. When the number of obstacles or moving objects increases in a dynamic environment, the extra computation time for detecting the interference of the additional objects will make the completion of the collision detection at the preset trajectory update rate impossible. In addition, if a collision of a moving object is detected, the motion of the object cannot be stopped or altered immediately. Therefore, a safety buffer space is usually added to the boundary of the object. The optimal size for this buffer is difficult to decide, which depends on the speed of the moving object. If the safety buffer is too large, a collision-free space will not be effectively used. On the other hand, if the safety buffer is too small, the collision cannot be avoided, which makes the collision detection system useless. Along this line, Foisy et al (1990) investigated the effect of the speeds and accelerations of moving objects on the imminent collision prediction. In general, the multiple interference detection method cannot be applied to detect collisions of fast moving objects in a dynamically varying environment. This is also true for a look-up table based collision detection method (Yu and Khalil, 1986), in which the collision-free space is mapped into the discretized joint space and stored in a table structure.

In the swept volume interference detection, the volumes swept out by moving objects are generated first. The interference detection between these swept volumes is then performed. The major challenge of this method is to represent the swept volumes of complicated moving objects in a computing system efficiently. Note that the collision detection by the interference detection of the swept volumes is not a necessary test because the moving objects may occupy the same space at different times. In addition, the exact time and location of the collision are very difficult to decide according to this method. Although some work along this line has been reported (Boyse, 1979; Cameron, 1985; Herman, 1986), taking the aforementioned drawbacks of this technique into account, its real-time application is not promising.

If when and where the collision will occur are concerned, the intersection detection between the moving objects modeled as four-dimensional entities (three spatial dimensions and time) needs to be performed. The technique used by Esterling and Rosendale (1983) and Cameron (1990) is based upon the extrusion of objects. The extrusion method is applicable to simple shapes and motions, but, it is difficult to detect the intersection of complicated moving objects. For example, the motions of objects implemented in (Cameron, 1990) is restricted to the translation motion only. Based upon the configuration space approach and quaternion representation of rotations, Canny (1986) derived an algorithm which can detect the exact collision points for a moving object that is simultaneously translating and rotating among obstacles. In Canny's implementation, the moving object is assumed to rotate in a nearly constant velocity while the center point of the object is traveling along a straight-line. The angular acceleration at the beginning and deceleration at the end of a motion segment has not been considered. In general, the path planning involving rotations in the configuration space is difficult, especially for the motion coordination of a robot manipulator with complicated kinematic chains such as the UPSarm which has ten actuators and consists of both serial and closed kinematic chains. The intersection detection algorithm developed by Kawabe et al (1988) can also compute the collision points between moving objects. However, in their implementation, the trajectories of moving objects are distorted due to the approximation of every element of the homogeneous motion transformation matrix by a cubic function of time. The scheme they proposed for controlling the level of approximation works in their simulation environment, but, it is not applicable for the real-time implementation.

FIGURE 1: The mechanical configuration of the UPSarm.

The intersection algorithm presented in this paper can calculate the exact time and location of the intersection between moving objects in real time without any approximation of motions. Motions of objects can be specified as point-to-point and/or continuous-path motions in the actuator space, effective joint space, or Cartesian space of a robot manipulator (Cheng, 1994). In addition, dimensions of moving objects for collision detection can change dynamically during the motion. Our intersection detection approach is designed for real-time collision detection when multiple robot arms are working in a clustered environment with a shared workspace. The intention is to provide a safety checking capability for manipulation of the high-speed UPSarm, especially when multiple robot arms load packages simultaneously. The prototype UPSarm shown in Figure 1 is developed to study the feasibility of loading packages inside a truck. The formulations for transformation of the kinematic information presented in actuator space, effective joint space, and world Cartesian coordinate space as well as real-time manipulation of the UPSarm have been reported in (Cheng, 1994). This paper describes the collision detection technique developed for this hybrid serial-and-parallel-driven redundant robot arm.

The rest of the paper is arranged as follows. We will first formulate general motions of moving objects in the s-domain so that the spatial information of the motion can be separated from the time aspect. Next, we present efficient intersection detection formulations for moving polyhedra and two rectangular solids. We then describe the real-time implementation of the intersection detection technique in a distributed computing environment. Finally, we illustrate the real-time capabilities and special features of our collision detection method by manipulating the UPSarm in an environment with an obstacle. The paper will be concluded with discussions of the results.

FORMULATION OF MOTION

Essential to our real-time intersection detection approach is to formulate complicated robot motions in the s-domain so that the collision detection problem in four-dimensional space can be solved in real time. The s-function is expressed in terms of time t. The spatial information will be separated from the time aspect under the s-formulation, then the intersection detection becomes *a geometrical problem.* In the following presentation, the motion of objects will be formulated in the s-domain in the framework of robot path planning and trajectory generation as implemented in our control system for real-time manipulation of the redundant UPSarm (Cheng, 1994). Different motion profiles and various motions, and their combinations described in this section are needed for our practical robot application of loading packages.

Formulation in s-Domain

In each robot motion segment, the s-parameter is a normalized value in the range of $0 \le s \le 1$, except for a segment-blended continuous path which will be discussed soon. For a square wave acceleration motion profile, the s-function can be derived in terms of time t as follows.

$$s = \begin{cases} \dfrac{t^2}{t_a(t_a+2t_s+t_d)}, & \text{acceleration } 0 \le t < t_a \\ \dfrac{2t-t_a}{t_a+2t_s+t_d}, & \text{slew motion } t_a \le t \le t_a+t_s \\ 1-\dfrac{(t_a+t_s+t_d-t)^2}{t_d(t_a+2t_s+t_d)}, & \text{deceleration } t_a+t_s < t \le t_a+t_s+t_d \end{cases} \tag{1}$$

where the acceleration time t_a, slew motion time t_s, and deceleration time t_d are calculated from the pre-specified maximum acceleration $a_{\max}$, deceleration $a_{\min}$, and velocity $v_{\max}$ of each actuator of a robot. These motion constraints are directly related to the available motor torque of each actuator. The values of t_a, t_s, and t_d for each actuator are computed by the following formulas.

$$t_a = \frac{n_v v_{\max}}{n_a a_{\max}} \tag{2}$$

$$t_d = \frac{n_v v_{\max}}{n_d a_{\min}} \tag{3}$$

$$t_s = \frac{\triangle \mathbf{q} - \triangle \mathbf{q}_a - \triangle \mathbf{q}_d}{n_v v_{\max}} \tag{4}$$

where n_v, n_a, and n_d are programmable speed, acceleration, and deceleration factors, respectively. The acceleration distance $\triangle \mathbf{q}_a$ and deceleration distance $\triangle \mathbf{q}_d$ are calculated by

$$\triangle \mathbf{q}_a = \frac{n_v^2 v_{\max}^2}{2n_a a_{\max}} \tag{5}$$

$$\triangle \mathbf{q}_d = \frac{n_v^2 v_{\max}^2}{2n_d a_{\min}} \tag{6}$$

The total traveling distance $\triangle \mathbf{q}$ for each actuator is obtained from its starting position and specified target position of a motion segment. The maximum values of t_a, t_s, and t_d for each actuator calculated by Eqs(2), (3), and (4) will be used in the formulation of s-function in Eq(1).

For a sinusoidal motion profile, the s-function is formulated as follows

$$s = \begin{cases} \dfrac{t_a\left(\dfrac{t}{t_a} - \dfrac{1}{\pi}\sin\dfrac{\pi t}{t_a}\right)}{t_a+2t_s+t_d}, & \text{acceleration } 0 \le t < t_a \\ \dfrac{2t-t_a}{t_a+2t_s+t_d}, & \text{slew motion } t_a \le t \le t_a+t_s \\ \dfrac{t+t_s+\dfrac{t_d}{\pi}\sin\dfrac{\pi(t-t_a-t_s)}{t_d}}{t_a+2t_s+t_d}, & \text{deceleration } t_a+t_s < t \le t_a+t_s+t_d \end{cases} \tag{7}$$

where the slew motion time t_s can also be computed by Eq(4), and acceleration time t_a and deceleration time t_d are calculated by

$$t_a = \frac{\pi n_v v_{\max}}{2n_a a_{\max}} \tag{8}$$

$$t_d = \frac{\pi n_v v_{\max}}{2n_a a_{\min}} \tag{9}$$

No matter whether the acceleration of a motion segment is a square-wave or sinusoidal profile, the s-value is always in the range of [0,1]. The value of s is 0 and 1 at the beginning and target positions of a motion segment, respectively. If the time t is known for a motion segment, the s-value can be readily calculated by Eq(1) or (7). In our intersection detection algorithm, the position where the intersection occurs is expressed in terms of a s-value. If the exact time t when the intersection occurs is needed, it can be derived as follows. If the value of s is less than $\frac{t_a}{t_a+2t_s+t_d}$, the intersection occurs during the acceleration phase of a motion segment so that the first equation of Eq(1) or (7) will be used to compute the time t corresponding to a given s-value. If the intersection occurs during the slew motion period, i.e. $s < \frac{t_a+2t_s}{t_a+2t_s+t_d}$, the second equation of Eq(1) will be invoked for the computation of t. Otherwise, the last equation of Eq(1) or (7) for deceleration will be used. The Newton-Raphson numerical method is used to solve for t with a given s-value when the first or third equation of Eq(7) is called for. The selected initial guess for t in this iterative numerical technique is 0 and $(t_a + t_s)$ for the first and third equations of Eq(7), respectively.

Point-to-Point Motion

In a point-to-point motion, we only concern the initial and final target positions of the robot arm. The position and orientation of any link of the robot manipulator can be obtained by using the transformation matrix

$$\mathbf{T}(s) = \mathbf{T}(\phi(s)) \tag{10}$$

where the joint values ϕ may be functions of the actuator values $\mathbf{q}(s)$ for hybrid serial-and-parallel-driven robot manipulators such as the UPSarm. For most industrial robot arms, the actuator values $\mathbf{q}$ are the same as the joint values ϕ. The actuator values $\mathbf{q}(s)$ can be expressed in terms of s by

$$\mathbf{q}(s) = \mathbf{q}_i + s\triangle\mathbf{q} \tag{11}$$

where the s-function can be formulated using Eq(1) or Eq(7) for the square-wave or sinusoidal acceleration motion profile, respectively. The traveling distances $\triangle\mathbf{q}$ of each actuator are computed from its initial positions $\mathbf{q}_i$ and final positions $\mathbf{q}_f$ of a motion segment as

$$\triangle\mathbf{q} = \mathbf{q}_f - \mathbf{q}_i \tag{12}$$

Straight-Line Motion

In many applications such as robot path tracking, the motion of the robot may be constrained in the Cartesian space. In this situation, the motion formulation in s-domain is different from the above point-to-point motion. As an example, the formulation for a straight-line motion will be addressed as follows. Other Cartesian motions can be treated in the same manner. In a straight-line motion, besides the computation of the acceleration time t_a, slew motion time t_s, and deceleration time t_d for each actuator of the robot, these times along a straight-line path are also calculated according to the total traveling distance

$$d = \sqrt{(x_{w,f} - x_{w,i})^2 + (y_{w,f} - y_{w,i})^2 + (z_{w,f} - z_{w,i})^2} \tag{13}$$

where $(x_{w,i}, y_{w,i}, z_{w,i})$ and $(x_{w,f}, y_{w,f}, z_{w,f})$ are the coordinates of the end points $\mathbf{P}_{w,i}$ and $\mathbf{P}_{w,f}$ of a straight-line in the world coordinate system $X_wY_wZ_wO_w$. The position of the object then can be calculated by

$$\mathbf{P}_w(s) = \mathbf{P}_{w,i} + (\mathbf{P}_{w,f} - \mathbf{P}_{w,i})s \tag{14}$$

where the s-function can be formulated using Eq(1) or Eq(7), depending on the profile of a motion segment. The actuator values $\mathbf{q}$ for the trajectory control can be obtained by the inverse and indirect kinematic solutions based upon the value of $\mathbf{P}_w(s)$.

If the orientations of the end-effector are not concerned while the TCP of the end-effector is moving along a prescribed path, the wrist actuator values can be computed by using the point-to-point motion and the effective joint values for the upper arm and forearm can be obtained according to the prescribed path for the TCP of the end-effector. This combined Cartesian-joint inverse kinematic solution for the UPSarm can be found in (Cheng et al, 1991).

While the TCP of the end-effector of the robot arm is moved along the prescribed path, the orientation of the end-effector can be smoothly changed from its beginning position to the final position, which is referred to as orientation interpolation. The orientation of the end-effector is interpolated by essentially using a two-rotation method, which involves two rotations in order to orient the end-effector from its initial orientation to the target orientation. The first rotation is about the common normal vector which normals to both approach vectors of the end-effector at the initial and final positions of the motion segment. The second rotation is about the approach vector itself from the initial position to the final position. These two rotation interpolations are carried out at each trajectory generation cycle simultaneously while the TCP of the end-effector is moving along the prescribed path. The formulas for these two orientations are given by

$$\beta(s) = \beta_i + (\beta_f - \beta_i)s \tag{15}$$

$$\gamma(s) = \gamma_i + (\gamma_f - \gamma_i)s \tag{16}$$

where the s-function can be formulated using Eq(1) or Eq(7) for the square-wave or sinusoidal acceleration motion profile; β_f and β_i are the rotation angles about the common normal, and γ_f and γ_i are the rotation angles about the approach vector at the beginning and end of a straight-line motion segment; respectively.

Segment-Blended Continuous Path

Two point-to-point motion segments can be blended to form a continuous path. The following cubic polynomial is implemented as a transition trajectory, which is continuous in both position and velocity.

$$\mathbf{q}(s) = \mathbf{b}_0 + \mathbf{b}_1 s + \mathbf{b}_2 s^2 + \mathbf{b}_3 s^3 \tag{17}$$

The s function at the transition segment is very simple.

$$s = \frac{t}{t_b} \tag{18}$$

where t_b equals the deceleration time t_d of the current segment or the acceleration time t_a of the next segment, whichever is larger. The value of t in Eq(18) varies from 0 to t_b so that $s \in [0, 1]$. The coefficients $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$, and $\mathbf{b}_3$ can be computed from the boundary conditions of the transition segment.

$$\mathbf{b}_0 = \mathbf{q}_i$$

FIGURE 2: Two types of object-obstacle intersection.

$$
\begin{aligned}
\mathbf{b}_1 &= \dot{\mathbf{q}}_i \\
\mathbf{b}_2 &= 3(\mathbf{q}_f - \mathbf{q}_i) - 2\dot{\mathbf{q}}_i - \dot{\mathbf{q}}_f \\
\mathbf{b}_3 &= -2(\mathbf{q}_f - \mathbf{q}_i) + \dot{\mathbf{q}}_i + \dot{\mathbf{q}}_f
\end{aligned}
$$

where $\mathbf{q}_i$ and $\dot{\mathbf{q}}_i$ are the actuator position and velocity at the end of the slew motion of the current segement, and $\mathbf{q}_f$ and $\dot{\mathbf{q}}_f$ are the actuator position and velocity at the beginning of the slew motion of the next segment. For blending two straight-lines, the actuator vector $\mathbf{q}$ will be replaced by the translational position vector $\mathbf{P}$. A blending trajectory using higher order polynomials can be formulated in the same manner. For example, a fifth order polynomial will include accelerations at the boundaries of the transition trajectory. When the current segment is blended with the next segment, the range of the s-vale in the current segment will be changed from $[0,1]$ to $[0, \frac{t_a+2t_s}{t_a+2t_s+t_d}]$.

FORMULATION OF INTERSECTION

We classify intersection as type-A and type-B intersections as is shown in Figure 2. The type-A intersection occurs when a vertex of a body touches a surface of other body. In our object-obstacle scenario, we further divide the type-A intersection into two sub-classes. The type-A1 intersection occurs when a vertex of the obstacle contacts a surface of the object; and type-A2 intersection occurs when a vertex of the object touches a surface of the obstacle. If an edge of the object contacts an edge of the obstacle, it is referred to as the type-B intersection. All other forms of contact are special cases which may be combination of more than one type of intersections.

In our current robot application, the packages handled by the UPSarm are restricted to rectangular packages only. The links of the UPSarm which are prone to collision with stacked packages are also in rectangular shapes as is shown in Figure 1. Therefore, the rectangular solid modeling for objects is a natural choice for the collision detection in our practical application. The rectangular solid is one of the most simple geometrical primitives, it has wide applications in the geometrical modeling of objects. Although other geometrical primitives may be more efficient for some applications, in general, any object can be modeled by a set of finite rectangular solids with a certain resolution. Therefore, if one can detect collision of two rectangular solids, the collision of two arbitrary-shaped objects which can be modeled as two sets of finite rectangular solids may be consequently detected. Hence, the collision detection for rectangular solids is addressed in details in this paper for our practical application purpose. Nevertheless, our motion formulation in the s-domain based upon a general-purpose path planner and trajectory generator is quite general, it can be used to control any mechanical devices. The location of any point of robot links, regardless of the shape of the link, can be determined at any point of time based upon the s-parameter. Therefore, for the generality, formulations for the intersection detection of moving polyhedra are also derived in this section.

Type-A Intersection

Type-A Intersection of Two Rectangular Solids. For the clarity of presentation, two rectangular solids under collision detection are identified as *object* and *obstacle* as is shown in Figure 3. In Figure 3, there are three coordinate systems which are the local object coordinate system $X_u Y_u Z_u O_u$, local obstacle coordinate system $X_v Y_v Z_v O_v$, and world coordinate system $X_w Y_w Z_w O_w$. In general, the coordinates for vertices U_i and V_i of the object and obstacle can be conveniently expressed in their corresponding local coordinate systems as is shown in Figure 3 for vertices U_1, U_7, V_1, and V_7. The information presented in the local coordinate systems can be reconciled in the world coordinate system. Figure 3 also shows twelve edges L_k of the object and six surfaces of the obstacle, which are used for the convenience of presentation. In this paper, the surface of S_i refers to a finite surface of the object or obstacle. The numbering system for surfaces of both object and obstacle shown in Figure 3 is the same. A plane is defined as an infinite plane which contains a surface. According to these definitions, surface S_i is a subset of plane S_i.

In type-A1 intersection detection of two rectangular solids, we will check if a vertex of the obstacle contacts a surface of the object in the local object coordinate space. There are six vertices for a rectangular obstacle and six surfaces for a rectangular object. Therefore, in general, we need to check thirty-six pairs of vertex and surface for type-A1 intersection detection. Since these testings are similar, only the intersection between vertex V_i of the obstacle and surface S_1 of the object will be considered. If the following conditions are satisfied at any point of the time, vertex V_i will contact surface S_1 of the object.

$$x_{u0}(t) - x_{u,i}(t) = 0 \tag{19}$$

$$y_{u0}(t) \leq y_{u,i}(t) \leq y_{ul}(t) \tag{20}$$

$$z_{u0}(t) \leq z_{u,i}(t) \leq z_{ul}(t) \tag{21}$$

where $(x_{u0}(t), y_{u0}(t), z_{u0}(t))$ and $(x_{ul}(t), y_{ul}(t), z_{ul}(t))$ are the coordinates for vertices U_7 and U_1 of the object in the local coordinate system $X_u Y_u Z_u O_u$, respectively. Note that these vertices are functions of the time, which means

FIGURE 3: Object, obstacle, and three coordinate systems.

that dimensions of the rectangular solid object may vary during the collision detection process. The coordinate $(x_{u,i}(t), y_{u,i}(t), z_{u,i}(t))$ of vertex V_i in the coordinate system $X_u Y_u Z_u O_u$ can be obtained by

$$\mathbf{V}_{u,i}(t) = (\mathbf{T}_u^v(t))^{-1}\mathbf{V}_i(t) = (x_{u,i}(t), y_{u,i}(t), z_{u,i}(t))^T \quad (22)$$

where the transformation matrix $\mathbf{T}_u^v(t)$, which will transform a point expressed in the local object coordinate system to a point in the obstacle coordinate system, is defined as

$$\mathbf{T}_u^v(t) = \mathbf{T}_w^v(t)\mathbf{T}_u^w(t) \quad (23)$$

The coordinate transformation matrices $\mathbf{T}_w^v(t)$ and $\mathbf{T}_u^w(t)$ are shown in Figure 3. Like the object, vertex $\mathbf{V}_i(t)$ of the obstacle is a function of time in the local coordinate system, which implies that dimensions of the obstacle may also change during the motion. Using the above formulation, the type-A1 intersection detection problem can be stated as: find the minimum time t which satisfies Eq(19) under the constraints (20) and (21). The major difficulty in solving this problem is to represent complicated motions of the object and obstacle analytically. To simplify the analysis, we have formulated the motion in the s-domain. The motion transformation matrix $\mathbf{T}_u^v(t)$ becomes $\mathbf{T}_u^v(s)$ with $s \in [0, s_f]$ and $0 < s_f \leq 1$. Normally, the value of s_f is 1 except for a segment-blended continuous path. With this additional constraint and reformulation of motion in the s-domain, Eqs(19), (20), and (21) become

$$x_{u0}(s) - x_{u,i}(s) = 0 \quad (24)$$
$$y_{u0}(s) \leq y_{u,i}(s) \leq y_{ul}(s) \quad (25)$$
$$z_{u0}(s) \leq z_{u,i}(s) \leq z_{ul}(s) \quad (26)$$
$$0 \leq s \leq s_f \quad (27)$$

The type-A1 intersection detection then becomes to solve the nonlinear Eq(24) under the constraints (25-27). There are various methods for solving this kind of constrained nonlinear equation. We choose to solve this system of nonlinear equation and inequalities through a modified bisection numerical method . The algorithm can be described as follows. Let

$$F(s) = x_{u0}(s) - x_{u,i}(s) \quad (28)$$

we first evaluate $F(0)$ and $F(s_f)$ at the beginning and end of a motion segment, respectively. If $F(0)$ and $F(s_f)$ carry different signs, there would be possible collisions during the motion of this segment. Then we calculate $F(s_f/2)$, and one of boundary points which has the same sign of $F(s_f/2)$ will be replaced by $s_f/2$ for the next iteration of bisection. This process continues until $F(s)$ reaches the preset tolerance. However, it is possible that there is still a collision during the motion even the signs of $F(0)$ and $F(s_f)$ are the same. To detect this kind of collision, we will evaluate $F(s)$ at different points of s within the range of $0 \leq s \leq s_f$ to see if there is a sign change. Once a sign change has been detected, which indicates a possible collision, the above described bisection method will be employed to find the value s at which $F(s)$ equals zero. The number of evaluation points can be determined dynamically, depending upon the traveling distance of each motion segment. The larger is the motion range, the more points shall be evaluated. However, this number of evaluation points is independent of the speed and motion profiles of each segment as it will be demonstrated later. If no solution exists for Eq(24) with $s \in [0, s_f]$, there is no type-A1 intersection. Otherwise, the s solutions to Eq(24) under the constraint (27) are then used to calculate $y_{u,i}(s)$ and $z_{u,i}(s)$. For a such given s-value, if $y_{u,i}(s)$ and $z_{u,i}(s)$ satisfy constraints (25) and (26), this s-value corresponds to a type-A1 intersection. Finally, the time t at which the first type-A1 intersection occurs will be obtained using the smallest s-value from the solution set of s-values which satisfy the nonlinear Eq(26) as well as constraint conditions (24), (25), and (26). We have described the computation of t from a given s-value. Note that if the time at which the collision would occur is not concerned, the last step can be omitted. Furthermore, the intersection of objects detected by our algorithm is the first occurrence of all possible collisions. Therefore, interference detections such as if a vertex is located inside a face are not necessary.

The type-A2 intersection can be detected in the same manner. For example, the intersection between vertex U_i of the object and surface S_1 of the obstacle can be detected by the following conditions

$$x_{v0}(s) - x_{v,i}(s) = 0 \quad (29)$$

$$y_{v0}(s) \leq y_{v,i}(s) \leq y_{vl}(s) \quad (30)$$

$$z_{v0}(s) \leq z_{v,i}(s) \leq z_{vl}(s) \quad (31)$$

$$0 \leq s \leq s_f \quad (32)$$

where $(x_{v0}(s), y_{v0}(s), z_{v0}(s))$ and $(x_{vl}(s), y_{vl}(s), z_{vl}(s))$ are the coordinates for vertices V_7 and V_1 of the obstacle, respectively; and $(x_{v,i}(s), y_{v,i}(s), z_{v,i}(s))$ is the coordinate of vertex $U_{v,i}$ expressed in the obstacle coordinate system $X_vY_vZ_vO_v$. The coordinate of vertex $U_{v,i}$ can be calculated by

$$\mathbf{U}_{v,i}(s) = \mathbf{T}_u^v(s)\mathbf{U}_i(s) = (x_{v,i}(s), y_{v,i}(s), z_{v,i}(s))^T \quad (33)$$

Type-A Intersection of Polyhedra. If the object is represented as a polyhedron whose surfaces are surrounded by straight-lines, the type-A1 intersection can be detected in the object coordinate system as follows. The equation of the plane which contains a surface of the object can be formulated as

$$\mathbf{L}_i(t)u + \mathbf{L}_j(t)v + \mathbf{U}_k(t) \quad (34)$$

where $\mathbf{L}_i(t)$ and $\mathbf{L}_j(t)$ are two vectors, and $\mathbf{U}_k(t)$ is a vertex on the surface. Vector $\mathbf{L}_k(t)$ can be obtained from vertices U_i and U_j. When the vertex $\mathbf{V}_{u,i}(t)$ of the obstacle touches the surface, the following relation can be established

$$\mathbf{V}_{u,i}(t) = \mathbf{L}_i(t)u + \mathbf{L}_j(t)v + \mathbf{U}_k(t) \quad (35)$$

Eq(35) can be rearranged in the matrix form

$$\begin{pmatrix} \mathbf{L}_i(t) & \mathbf{L}_j(t) & \mathbf{U}_k(t) - \mathbf{V}_{u,i}(t) \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = 0 \quad (36)$$

If the above matrix equation has nonzero solutions, the determinant equation

$$|\mathbf{L}_i(t) \quad \mathbf{L}_j(t) \quad \mathbf{U}_k(t) - \mathbf{V}_{u,i}(t)| = 0 \quad (37)$$

must be satisfied. Formulate the motion in the s-domain, Eq(37) becomes

$$|\mathbf{L}_i(s) \quad \mathbf{L}_j(s) \quad \mathbf{U}_k(s) - \mathbf{V}_{u,i}(s)| = 0 \quad (38)$$

Similar to the previous section, by solving Eq(38) in the range of $0 \leq s \leq s_f$, the possible collision locations between the vertex and plane are obtained. The conclusion of the intersection will be confirmed by checking if any intersection point is located inside the boundary of the surface. This can be done by checking the sign of the scalar product of the the vector, connecting the intersection point with a vertex of of the polygon, and the normal vectors of its adjacent two edges. This type of equation is solved for all the pairs between the vertices of the obstacle and surfaces of the object. The type-A2 intersection detection for an obstacle modeled as a polyhedron can be performed in the same manner.

Type-B Intersection

If an edge of the object contacts an edge of the obstacle, it is referred to as the type-B intersection. The type-B intersection detection for both rectangular solids and polyhedra is the same, which can be performed in one of three coordinate spaces: object, obstacle, and world. In the following derivation, we choose the obstacle coordinate space. The line equations for edge U_iU_j of the object and edge V_iV_j of the obstacle can be formulated as

$$\mathbf{T}_u^v(t)(\mathbf{U}_j(t) - \mathbf{U}_i(t))u + \mathbf{T}_u^v(t)\mathbf{U}_i(t) \quad (39)$$

$$(\mathbf{V}_j(t) - \mathbf{V}_i(t))v + \mathbf{V}_i(t) \quad (40)$$

where paramters u and v are in the range of [0,1] at a given time. If two edges U_iU_j and V_iV_j intersect, the following equation holds

$$\mathbf{T}_u^v(t)(\mathbf{U}_j(t) - \mathbf{U}_i(t))u + \mathbf{T}_u^v(t)\mathbf{U}_i(t) = (\mathbf{V}_j(t) - \mathbf{V}_i(t))v + \mathbf{V}_i(t) \quad (41)$$

We can rearrange Eq(41) in the following matrix form

$$\begin{pmatrix} \mathbf{T}_u^v(t)(\mathbf{U}_j(t) - \mathbf{U}_i(t)) & \mathbf{V}_i(t) - \mathbf{V}_j(t) & \mathbf{T}_u^v(t)\mathbf{U}_i(t) - \mathbf{V}_i(t) \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = 0 \quad (42)$$

If this matrix has solutions, i.e. lines U_iU_j and V_iV_j contact each other, the determinant equation

$$|\mathbf{T}_u^v(t)(\mathbf{U}_j(t) - \mathbf{U}_i(t)) \quad \mathbf{V}_i(t) - \mathbf{V}_j(t) \quad \mathbf{T}_u^v(t)\mathbf{U}_i(t) - \mathbf{V}_i(t)| = 0 \tag{43}$$

must be satisfied. Like the type-A intersection detection, we also transfer the motion from the time t-domain to the parameter s-domain. Eq(43) then becomes

$$|\mathbf{T}_u^v(s)(\mathbf{U}_j(s) - \mathbf{U}_i(s)) \quad \mathbf{V}_i(s) - \mathbf{V}_j(s) \quad \mathbf{T}_u^v(s)\mathbf{U}_i(s) - \mathbf{V}_i(s)| = 0 \tag{44}$$

Similar to the type-A intersection detection, by solving this equation for $s \in [0, s_f]$ using the bisection method, we obtain the answers that may be the points where type-B intersections occur. For every solution of s, we solve two of three equations in Eq(42) for parameters u and v. If both values of u and v are in the range of [0, 1], the intersection point falls on edges of the object and obstacle. Otherwise, no intersection occurs. This detection procedure needs to be performed for each pair of edges of the object and obstacle. The smallest value of s from these detection procedures corresponds to the time at which the first type-B intersection occurs. Using this s-value, the exact collision time of the type-B intersection then can be obtained according to the method described before.

REAL-TIME IMPLEMENTATION

This section describes how a collision detection module is integrated into a real-time robot manipulation system. Although the real-time implementation presented in this section is specifically for the manipulation of the hybrid serial-and-parallel-driven redundant UPSarm, the ideas are general and can be applied to real-time collision detection of any mechanical devices. In our collision detection implementation, we will be able to manipulate the UPSarm by our robot language in the usual manner. But, whenever there is an imminent collision detected by the intersection detection algorithm, the robot arm will come to stop and the system will report the exact time at which the potential collision would occur. The intersection detection is performed at the path planning phase of a motion segment because the information such as acceleration time t_a, slew motion time t_s, and deceleration time t_d are available at this stage. In the context of this paper, the path planning refers to the process which calculates the required parameters, such as duration times of acceleration, slew motion, and deceleration, etc. of motion segments based on certain physical constraints. Taking the collision avoidance into account, the path planning will also detect the possible collision of each motion segment based upon the physical dimensions of each link of a robot and obstacles in the environment. The trajectory generation is a process which computes the intermediate positions of the robot arm in a motion segment based upon the information obtained by the path planner.

The electrical hardware for real-time motion coordination of the UPSarm is shown in Figure 4, which is a VMEbus-based real-time distributed computing system.

FIGURE 4: Real-time distributed computing system.

There are five VME boards in an industrial standard VME chassis. The Motorola MVME133 board is the target host running a MC68020 microprocessor with clock speed 16.67 MHz and 1Mb dynamic RAM. This target board is connected to an Ethernet network. The program for path planning and trajectory generation written in the C programming language is cross-compiled on a Sun SPARCstation and then down loaded to the target board via the Ethernet connection. The user interface with the robot arm is accomplished through two RS-232 serial communication ports of the target board. One serial communication port is connected to the SPARCstation for keyboard input of our robot language instructions and the other is connected to a teach pendant. Four Galil DMC530 VME servo control boards are responsible for servo control and low level communication of the control system. The motion command generated from the target board is sent to these servo boards via the VMEbus. Each servo board controls three brushless DC motors. The PID closed-loop control algorithm executed on a MC68008 microprocessor controls motion of each brushless DC motor. The sampling time interval T_s of the PID servo control loop is 1 ms.

Path planning task and trajectory generation task are two major tasks executed on the target board under real-time operating system pSOS$^+$. The system real-time clock with preset time unit T_c of 1 ms enables the clock interrupt service routine to keep track of the system cycle time. A priority-based preemptive algorithm has been used for real-time task scheduling, which means that at any point in time, the running task is the one with highest priority among all ready-to-run tasks. If a task, which has higher priority than the current running task, is ready to run, it will take over the CPU resource and the current running task will be swapped out. In our implementation, the path plan-

FIGURE 5: The CPU resource distribution.

ning task is created with lower priority than the trajectory task. The distribution of the CPU time of the target board is shown in Figure 5. The clock interrupt service routine is interrupted by the real-time clock every T_c ms. At every trajectory update cycle time T_t, which is 32 ms in our current implementation, the clock interrupt service routine will resume the suspended trajectory generation task. At the beginning of each trajectory update cycle, the trajectory task resumed by the clock interrupt service routine in the target board will send a set of motor encoder counts, which are linear functions of actuator values $\mathbf{q}$, and feedforward torques in DAC counts to servo boards via the VMEbus. After communication with servo boards, the trajectory task will increment the time tick and calculate the new set of actuator values and feedforward torques for the next position of the trajectory. After these activities, the trajectory task will suspend itself so that it will not contend for the CPU time. Once the trajectory task has been suspended, the planning task which has lower priority will become the running task and take over the CPU time of the target board. The path planning task will execute our robot language instructions if the user has typed in any command on the keyboard. For a motion statement, the path planning task will plan the path for the next motion segment based upon the target positions of the current and next segments along with physical constraints of the system. After the computation of the acceleration time t_a, slew motion time t_s, and deceleration time t_d, etc., the planning task will detect if there is intersection between the robot arm and the obstacle for an entire motion segment using the intersection detection algorithm presented in the previous section. If there will be a intersection, the minimum s-value and its corresponding time in the motion segment will be reported; the robot will be brought to stop immediately, which is achieved by changing the normal trajectory generation routine to a special stop routine for the execution in the trajectory generation task. In the meantime, the real-time clock keeps interrupting the clock interrupt service routine which is responsible for maintaining the trajectory cycle time. As discussed before, the clock interrupt service routine will resume the suspended trajectory task at every T_t ms. Since the trajectory task has higher task priority than the path planning task, the real-time operating system will swap out the path planning task when a task with higher priority is ready to run. The trajectory task will start running right after it is resumed by the clock interrupt service routine. The next trajectory update cycle will start as is shown in Figure 5.

Since the clock interrupt service routine is invoked every clock tick of the system, it has been kept at the briefest form possible. At the beginning of every trajectory update cycle, a new set of encoder counts and feedforward torques is sent to servo boards. The computation of this information in the trajectory task must be completed within the trajectory update cycle time T_t. If for some reason, it could not finish these calculations, a new set of encoder counts will not be available to the servo controller, which would result in a non-smooth motion trajectory and the desired path of the robot arm, say a straight-line, will not be achieved. Therefore, the trajectory task has also been kept as simple as possible in our software design and will guarantee the completion of the computations for the next set of encoder counts and feedforward torques. The path planning task has a lowest priority, it will consume all the remaining CPU time of the system. Hence, the computational expensive intersection detection algorithm are carried out in this low-priority task. The path planning task will be swapped out at the beginning of every trajectory update cycle. After the trajectory task is suspended, it will gain the CPU resource and restart the execution at the place where the task was previously stopped. If the planning task has insufficient time to run such as in case of blending two very short motion segments, it is likely that a continuous-path will be broken up as point-to-point motions. However, for each point-to-point motion segment, the achieved trajectory is always smooth and free of collision. Note that, since the collision detection is performed at the path planning stage and before the start of the motion segment, there is sufficient time for a robot to make a wise decision about the next movement to avoid a collision. The collision avoidance strategy of a robot is dictated by each robot task, therefore, the detailed exploration of general collision avoidance is beyond the scope of this paper.

EXPERIMENTATION

Testing Environment

The capabilities and special features of our intersection detection technique are illustrated by manipulation of the UPSarm in an environment with an obstacle. Figure 1 shows the translational actuator variables of the UPSarm in its actuator space. Both shoulder and wrist subsystems of the manipulator are designed with closed kinematic chains primarily due to considerations of rigidity and high payload-carrying capacity of the manipulator. To facilitate the kinematic analysis of this hybrid serial-and-parallel-driven redundant robot manipulator, three coordinate spaces: the world Cartesian coordinate space, effective joint space, and actuator space are introduced. The manipulator end-effector can be represented in one of the aforementioned three spaces. The 4x4 homogeneous trans-

FIGURE 6: Moving objects and obstacle.

formation matrix defines the TCP and orientation of the end-effector in the world Cartesian coordinate space by the 3x1 translational vector and 3x3 orientational submatrix, respectively. The effective joint space is used to represent the manipulator in an effective serial kinematic chain. An effective joint may not be a physical joint and is a virtual joint such as the second effective joint of the UPSarm. The actuator space locates the end-effector by specifying actuator angles or translational distances in actuator axes which may be directly connected to motor shaft axes. Unlike an effective joint, the defined actuator joint always physically exists. The actuator value is a linear function of the encoder count of a motor encoder mounted on a motor shaft. In this paper, the actuator values and effective joint values of the UPSarm are represented by $\mathbf{q} = [q_1, q_2, ..., q_{10}]^T$ and $\phi = [\phi_1, \phi_2, ..., \phi_{10}]^T$, respectively. Figure 6 gives the location of all actuators of the UPSarm. Figure 7 is the equivalent kinematic representation of the UPSarm in an effective joint space. In this effective joint space, the UPSarm is modeled as a ten degree-of-freedom serial kinematic chain. When the manipulator is represented in this effective joint space, the D-H notation (Denavit-Hartenberg, 1955) can be used to parameterize the geometrical information of the UPSarm. According to the D-H notation definition, the coordinate systems used for motion coordination of the UPSarm are shown in Figure 7, where $X_bY_bZ_bO_b$ is the base coordinate system of the arm.

In our testing environment, the world coordinate system $X_wY_wZ_wO_w$ is fixed in the corner of a trailer as is shown in Figure 6. The obstacle in the workspace of the UPSarm is a rectangular box depicted in dashed lines. The coordinates of the vertices V_1 and V_7, shown in Figure 3, of this obstacle

FIGURE 7: The UPSarm in effective joint space.

FIGURE 8: The variable bounding box and small boxes of the UPSarm.

in the world coordinate system $X_wY_wZ_wO_w$ are (75 in, 30 in, 50 in) and (50 in, 5 in, 0), respectively. The translational position vector (b_x, b_y, b_z) of the transformation matrix $\mathbf{T}_b^w$ is (126 in, 27 in, 80 in). The obstacle coordinate system $X_vY_vZ_vO_v$ in this case coincides with the world coordinate system $X_wY_wZ_wO_w$, which means that the transformation matrix $\mathbf{T}_w^v$ is an identity matrix. Only links from body 4 up to the end-effector of the UPSarm are prone to collision with the obstacle. The variable bounding box which encloses these links are shown in Figure 8. For efficiency, some impossible type-A and type-B intersection scenarios are excluded from the detection process in our implementation. If we treat the bounding box of the UPSarm as the object in the object-obstacle scenario addressed in the previous section, the local object coordinate system $X_uY_uZ_uO_u$ is the body coordinate system $X_4Y_4Z_4O_4$ shown in Figure 7. The transformation matrix $\mathbf{T}_u^v$ in Eq(23) becomes

$$\mathbf{T}_u^v = \mathbf{T}_u^w = \mathbf{T}_4^w = \mathbf{T}_b^w\mathbf{T}_1^b\mathbf{T}_2^1\mathbf{T}_3^2\mathbf{T}_4^3$$
$$= \begin{bmatrix} -c\phi_2 c\phi_3 & -s\phi_2 & -c\phi_2 s\phi_3 & b_x - a_2 c\phi_2 \\ -s\phi_2 c\phi_3 & c\phi_2 & -s\phi_2 s\phi_3 & b_y - a_2 s\phi_2 + \phi_1 \\ -s\phi_3 & 0 & -c\phi_3 & b_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (45)$$

where $s\phi_i$ and $c\phi_{i-1}$ represent $\sin\phi_i$ and $\cos\phi_i$, respectively; (b_x, b_y, b_z) is the translational part of the transformation matrix $\mathbf{T}_b^w$. The motion of the UPSarm is generated in the s-domain as described before. For a point-to-point motion, the effective joint values ϕ in Eq(45) can be obtained from the actuator values $\mathbf{q}$ by the direct kinematics. For a straight-line motion, the effective joint values ϕ in Eq(45) can be calculated by the inverse kinematic solution. The coordinates of vertices U_1 and U_7, shown in Figures 3 and 8, of the variable bounding box in the body coordinate system $X_4Y_4Z_4O_4$ are $(3.5\,\text{in}, 5.3\,\text{in}, d_8 + q_4 + q_5 + q_6)$ and $(-3.5\,\text{in}, -5.3\,\text{in}, 0)$, respectively. Note that the coordinate of vertex U_1 is a function of the time even expressed in the local coordinate system, which implies that dimensions of the moving bounding box may change during the motion.

Motion Examples

The following three simple robot motion examples will demonstrate the real-time collision detection capability of the system. The first motion example moves joint 3 of the robot arm from its initial home position to 70 degrees by executing a single robot language command

```
move #3 70
```

The integral value following the symbol # is the joint number and the last argument in this command is the target value of the joint for this motion segment. The home position of the UPSarm is the position with the effective joint value $(-27, 0, 21.5, 0, 0, 0, 0, 90, 90, 97.5)$, corresponding to the actuator value $(-27, -27, 4.9, 0, 0, 0, 0, 90, 9.7, 9.7)$. In our prototype system, the default speed, acceleration, and deceleration for the robot motion are set to twenty percent of the maximum defined ones. The default acceleration and deceleration motions are in sinusoidal profiles. In the above motion, the default speed, acceleration, and deceleration factors, and default motion profiles for acceleration and deceleration are used. If there was no collision, joint 3 of the UPSarm would move from its initial effective joint value -21.5 degrees to the target position 70 degrees, which correspond to actuator values of 44.9 inches and 64.5 inches, respectively. The profiles of the actuator value (inch), velocity (inch/sec), and acceleration (inch/sec^2) of actuator 3 would be shown in Figure 9, where the solid line is the ac-

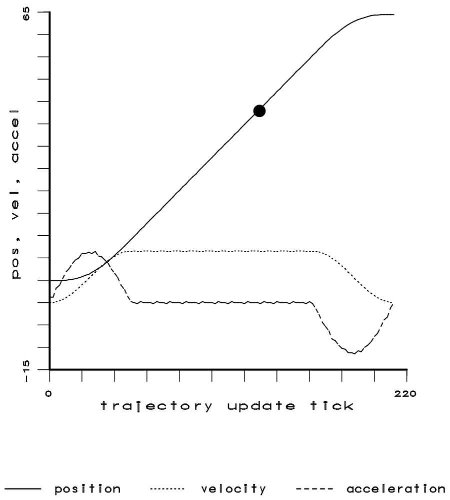

FIGURE 9: The motion profile of actuator 3 for the command `move #3 70`.

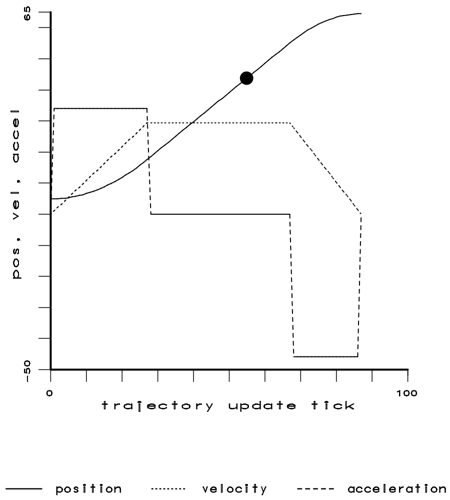

FIGURE 10: The motion profile of actuator 3 for the program in Listing 1.

tuator value with respect to trajectory update cycle tick, dotted line for linear velocity, and dashed line for linear acceleration. Each trajectory update tick corresponds to 32 ms in our current implementation. The acceleration, slew motion, and deceleration of this motion segment would take 50, 112, and 50 trajectory ticks, respectively. The total motion would take 6.784 seconds. However, when the above motion command is executed, the path planning task will report that there is a possible collision at the s-value of 0.6417 which corresponds to the effective joint value of 32.8 degrees or actuator value of 43.1 inches for actuator 3. With this s-value, and acceleration time t_a, slew motion time t_s, and deceleration time t_d, the exact time when the collision would occur is calculated by Eq(7). The possible collision would take place 4.127 seconds after the beginning of the motion during the slew motion period, which is at the 129th trajectory tick marked with a dark spot in Figure 9.

As discussed before that the s-formulation of the intersection detection algorithm separates the geometrical information of the moving objects from the time information. Under the s-formulation, the computational complexity of the intersection detection algorithm will no longer depend on the speed and motion profiles of moving objects. This result will be demonstrated by the second motion example. The robot motion program for this example is given in Listing 1.

LISTING 1

```
disable sinusoidalaccel
disable sinusoidaldecel
speed 0.5
acceleration 0.6
deceleration 0.8
move #3 70
```

The last statement of the program is the same as the first example, which will move effective joint 3 from its initial position to the target position of 70 degrees. The first two statements will disable the default sinusoidal acceleration and deceleration profiles so that the square-wave motion profile will be invoked. The subsequent three statements will change the default speed, acceleration, and deceleration to 50, 60, and 80 percent of the maximum ones, respectively. This is achieved by setting the variables n_s, n_a, and n_d in Eqs(2)-(9) to 0.5, 0.6, and 0.8, respectively. If there was no collision, actuator 3 would be moved according to the motion profile shown in Figure 10, where the solid line is the actuator value with respect to trajectory update cycle tick, dotted line for linear velocity, and dashed line for linear acceleration. The motion would take 27, 40, and 20 trajectory ticks for the acceleration, slew motion, and deceleration, respectively. The acceleration and deceleration times are different as is specified by the program in Listing 1. Like the first example, the path planning task will report that there will be a collision when the s-value is 0.6417, which corresponds to the time of 1.736 seconds after the beginning of the motion. The collision would occur during the slew motion at the 54th trajectory tick marked with a dark spot in Figure 10. Note that although the exact collision times for example one and example two are different, the s-values at which the intersections would occur are the same for both examples, which can be interpreted as that the collision location for both examples are the same. As a matter of fact, if the exact collision time is not concerned as in many applications, the intersection detection for the above two examples are the same. Based upon the initial and target positions, the intersection location corresponding to the s-value of 0.6417 for the above two examples can be detected with the same computational complexity, independent of the speed, acceleration, deceleration, and motion profiles of the motion segment.

As mentioned before that our intersection algorithm can handle moving objects which will change their dimensions during the motion. This feature is illustrated by the third motion example, which just replaces the last statement of the program in Listing 1 by the statement

```
move actuator(-27, -27, 64.5, 8,
              0, 0, 0, 90, 9.7, 9.7)
```

The second argument of the above command is the target position of the robot arm specified in the actuator space. Comparing the statement `move #3 70` with the above statement, the effective difference of motion is that not only actuator 3 is moved from its initial position to 64.5 inches, but also actuator 4 is extended 8 inches out from its initial zero position. The size of the variable bounding box of the UPSarm is getting bigger during the motion, but, the motion of actuator 3 and the acceleration, slew, and deceleration trajectory ticks are the same as those in example two. By executing the third motion program, the system will report that the collision would occur when the s-value is 0.600 at time t of 1.652 sec. which corresponds to the 52th trajectory cycle tick. Since the size of the bounding box increases during the motion in this example, the collision will occur sooner than that in the second example although both examples have the same motion profile, speed, and acceleration for the bounding box at any instant of the time.

CONCLUSIONS

A general-purpose collision detection technique for four-dimensional (three spatial dimension and time) objects has been presented in this paper. Our intersection detection technique can predict the exact collision time and location of moving objects in real time. For our practical application purpose, efficient formulations of intersection for two rectangular solids and polyhedra are also derived. The formulations for intersection detection between moving objects are based upon a general-purpose path planner and trajectory generator, which can be used for manipulation of any robot manipulator or mobile robot in point-to-point and continuous-path motions so long as an appropriate kinematics module is provided. Our collision detection algorithm separates time and motion characteristics from the geometrical information by using s-formulations, which enables the collision detection being performed in real time. Under the s-formulation, the intersection detection becomes a geometrical problem which is solved by a modified bisection method; the computational complexity of the intersection detection algorithm is then independent of the speed and motion profiles of moving objects, which is essential for its real-time application. Other novelty of this algorithm is that dimensions of moving objects can vary dynamically during the intersection detection process.

The real-time implementation of this algorithm and integration with a general-purpose path planner and trajectory generator for manipulation of the UPSarm, which is a ten degree-of-freedom hybrid serial-and-parallel-driven robot manipulator, in a distributed computing system have been described. The real-time collision detection capabilities of the algorithm for handling moving objects whose dimensions vary with time have been demonstrated by manipulating the UPSarm in an environment where an object presents. In our real-time implementation, due to the priority-based preemption, the trajectory generation task will dictate the performance of each programmed motion segment after a collision-free path is planned. Since the intersection detection is performed in the planning task with the lowest priority, the increase of the number of obstacles will not affect the performance of each programmed motion segment. Therefore, the algorithm is especially suitable for the collision detection of robots working in a dynamically varying environment. If we consider a package hold by the UPSarm as an additional moving object in our application examples, the transformation matrix $\mathbf{T}_u^v = \mathbf{T}_t^w$ between this moving object and the obstacle is available from the forward kinematics, and intersection detection between the package and the obstacle can be treated in the same manner as the variable bounding box with respect to the obstacle. Hence, the redundancy and closed-kinematic chains of the robot arm will not introduce any additional complexity from the intersection detection point of view as demonstrated by manipulation of the prototype UPSarm.

Although the obstacle in the presented motion examples is stationary, the intersection algorithm can be applied to the collision detection of multiple moving objects. For example, when two UPSarm operate synchronously with a common workspace, the constraint equations for the intersection can be described by a single s-parameter. By treating one arm as an object and the other as an obstacle, the collision detection of two robot arms can be readily performed by the intersection detection algorithm presented in this paper. If the working space is modeled by a region octree with each linked cubic volume leaf node being either completely filled by an obstacle or completely empty (Herman, 1986; Arimoto et al, 1988), the algorithm presented in this paper is also well suitable for detecting collision in this representation.

Acknowledgement

The major work described in this paper was done when the author was with Research and Development of United Parcel Service, Inc., and it was supported in part by the National Science Foundation as a Research Initiation Award under Grant DMII-9309207 and by a Junior Faculty Research Fellowship from University of California, Davis.

REFERENCES

Arimoto, S., Noborio, H., Fukuda, S. and Noda, A., A Feasible Approach to Automatic Planning of Collision-Free Robot Motions, *Robotics Research, The Fourth International Symposium*, (edited by R. C. Bolles and B. Roth), 1988, pp. 489-496.

Borenstein, J. and Koren, Y., Real-Time Obstacle Avoidance for Mobile Robots, *IEEE Trans. Systems, Man, and Cybernetics*, vol. SMC-19, no. 5, 1989, pp. 1179-1187.

Boyse, J. W., Interference Detection among Solids and Surfaces, *Commun. ACM*, Vol. 22, No. 1, 1979, pp. 3-9.

Brooks, R. A., Solving the Findpath Problem by Good Representation of Free Space, *IEEE Trans. Systems, Man, and Cybernetics*, vol. SMC-13, no. 3, 1983, pp. 190-197.

Cameron, S., A Study of the Clash Detection Problem in Robotics, *Proc. IEEE Int. Conf. Robotics and Automation*, St. Louis, MO, March 1985, pp. 488-493.

Cameron, S., Collision Detection by Four-Dimensional Intersection Testing, *IEEE Trans. on Robotics and Automation*, Vol. 6, No. 3, June, 1990, pp. 291-302.

Canny, J., Collision Detection for Moving Polyhedra, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 2, March 1986, pp. 200-209.

Cheng, H. H., Real-Time Four-Dimensional Collision Detection for Robot Manipulators, *Proc. of the 1st Chinese World Congress on Intelligent Control and Intelligent Automation*, Beijing, China, August 26-30, 1993, vol. 2, pp. 1244-1251, vol. 3, pp. 2223-2228.

Cheng, H. H., Real-Time Manipulation of a Hybrid Serial-and-Parallel-Driven Redundant Industrial Manipulator, *ASME Journal of Dynamic Systems, Measurement, and Control*, Vol. 116, December, 1994, pp. 687-701.

Cheng, H. H. and Gupta, K. C., Vectorization of Robot Inverse Dynamics on a Pipelined Vector Processor, *IEEE Trans. on Robotics and Automation*, Vol. 9, No. 6, December, 1993, pp. 858-863.

Cheng, H. H., Lee, J. J., and Penkar, R., Kinematics of the Prototype UPSarm, *Proc. of the 2nd National Applied Mechanisms and Robotics Conference*, Vol. II, Cincinnati, Ohio, November 3-6, 1991, pp. IXC.2-1 to IXC.2-11.

Cheng, H. H. and Gupta, K. C., A Study of Robot Inverse Kinematics Based upon the Solution of Differential Equations, *Journal of Robotic Systems*, Vol. 8, No. 2, April 1991, pp. 159-175.

Cheung, E. and Lumelsky, Motion Planning for a Whole-Sensitive Robot Arm Manipulator, *Proc. IEEE Int. Conf. Robotics Automation*, Cincinnati, Oh, 1990, pp. 344-349.

Denavit, J. and Hartenberg, R. S., A Kinematic Notation for Lower Pair Mechanisms Based on Matrices, *ASME Trans. J. Applied Mech.*, vol. 22, 1955, pp. 215-221.

Donald, B. R., A Search Algorithm for Motion Planning with Six Degrees of Freedom, *Artificial Intell. J.*, vol. 31, Mar. 1987, pp. 295-353.

Esterling, D. M. and Rosendale, J. V., An Intersection Algorithm for Moving Parts, *NASA Symposium on Computer-Aided Geometric Modeling*, April 20-22, 1983, pp. 119-123.

Foisy, A., Hayward, V., and Aubry, A., The Use of Awareness in Collision Prediction, *Proc. IEEE Int. Conf. Robotics Automation*, Cincinnati, Oh, 1990, pp. 338-343.

Herman, M., Fast, Three-Dimensional, Collision-Free Motion Planning, *Proc. IEEE Int. Conf. Robotics Automation*, San Francisco, Apr. 1986, pp. 1056-1063.

Khatib, O., Real-Time Obstacle Avoidance for Manipulators and Mobile Robots, *Int J. Robotics Res.*, Vol. 5, No. 1, 1986, pp. 90-98.

Kawabe S., Okano, A., and Shimada, K., Collision Detection among Moving Objects in Simulation, *Robotics Research, The Fourth International Symposium*, (edited by R. C. Bolles and B. Roth), 1988, pp. 489-496.

Kim, J. O. and Khosla, P. K., Real-Time Obstacle Avoidance Using Harmonic Potential Functions, *IEEE Trans. on Robotics and Automation*, Vol. 8, No. 3, June 1992, pp. 338-349.

Latombe, J. C., *Robot Motion Planning*, Kluwer Academic Publishers, 1991.

Maciejewski, A. and Klein, C., Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments, *Int J. Robotics Res.*, Vol. 4, No. 3, 1985, pp. 109-117.

Oh, S. Y., Orin, D., and Bach, M., Inverse Kinematic Solution for Kinematically Redundant Robot Manipulators, *J. Robotic Systems*, 1(1), 1984, pp. 235-249.

Lozano-Perez, T., Automatic Planning of Manipulator Transfer Movements, *IEEE Trans. Systems, Man, and Cybernetics*, vol. SMC-11, no. 10, 1981, pp. 681-698.

Lozano-Perez, T., Spatial Planning: A Configuration Space Approach, *IEEE Trans. Computers*, vol. C-32, no. 2, 1983, pp. 108-120.

Roach, J. W. and Boaz, M. N., Coordinating the Motions of Robot Arms in a Common Workspace, *IEEE Trans. on Robotics and Automation*, Vol. 3, No. 5, Oct. 1987, pp. 437-444.

Shaffer, C. A. and Herb, G. M., A Real-Time Robot Arm Collision Avoidance System, *IEEE Trans. on Robotics and Automation*, Vol. 8, No. 2, April 1992, pp. 149-160.

Yu, Z. and Khalil, W., Table Look up for Collision Detection and Safe Operation of Robots in *Theory of Robots*, (selected papers from IFAC/IFIP/IMACS symp., Vienna, Austria), Dec. 1986, pp. 343-347.