Computers Elect. Engng Vol. 17, No. 3, pp. 205–215, 1991
Printed in Great Britain.

0045-7906/91 $3.00 + 0.00

REAL-TIME ROBOT ARM COLLISION DETECTION FOR TELEROBOTICS

CLIFFORD A. SHAFFER
Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, U.S.A.

(Received 10 December 1990; accepted in final revised form 11 April 1991)

Abstract—The role of real-time collision detection as part of a safety system in telerobotics applications is discussed. We survey a number of proposed collision detection systems. These methods differ in hardware requirements, speed and the information available for use when responding to an immiment collision. In particular, we compare systems based on simulation of the robots' workspace with systems that react mainly to local sensor information (i.e. proximity sensors). Both approaches to collision detection are rapidly improving. Hierarchical data structures allow real-time simulation of the environment with standard microprocessors. Advances in proximity sensor hardware allow for greater information about the current state of the workspace. While at the moment most experimental systems tend to be entirely either simulation or sensor based, future systems are likely to use local sensor input to update a sophisticated model of the workspace. We also examine how precision in the representation vs collision detection time and robot arm stopping time affect the size of the safety buffer required by both approaches. Finally, the problems of gripping and response to imminent collision are addressed.

1. INTRODUCTION

This paper describes recent developments in real-time collision detection systems. Our primary interest in these methods is their application to a real-time safety system for multiple robot arms in both teleoperated and machine controlled environments. The methods vary in a number of ways, such as complexity of the system, time requirements (real-time vs off-line) and the amount of special-purpose hardware required. In particular, we recognize a distinction between methods that rely on hardware sensors to continuously gather local information about the environment (which we will term the *proximity sensor* method), as opposed to software simulation of movement within a predefined environment (which we will term the *simulation* method). The simulation method typically maintains a geometric world model describing the working environment.

We discuss certain unsettled issues in the implementation of a robot safety system. We examine how precision in representation, collision detection time and robot arm stopping time affect the size of the *safety buffer* required by the safety system. The safety buffer is that region around the robot arm that must remain clear of other objects. This safety buffer may change size with arm speed and direction of motion. We also discuss how differences in the information available to the two approaches (simulation vs sensor input) affects possible responses when faced with an imminent collision. Our third issue is how the collision detection module affects the operation of gripping objects by the robot's end effector.

Conceptually, the proximity sensor approach is quite simple. We mount a set of sensors at strategic locations on the robot arm. Whenever a sensor detects another object moving within its sensing range, the sensor can report the position of that object. If the distance falls below the safety buffer threshold, a collision is imminent. The simplicity of this approach (in its most primitive form) argues in favor of its adoption for a fail-safe, low-level safety system. Currently the major issue for the proximity sensor method is finding sensors that: (1) can detect objects regardless of their material composition or color; and (2) can detect objects at a suitable range [1]. Another issue is that purely sensor-based approaches (i.e. those that store no model of the workspace) have only local information about the environment, and thus can only make local decisions in response to imminent collision.

Simulation-based approaches attempt to model the robot arm workspace, along with all movement of objects within the workspace. At all times, the positions of all objects that can affect the arms must be known. In some applications we can assume that all objects other than the arms and objects gripped by the arms are static. In other applications, we will need to receive information

about the movement of objects, which may require some form of sensor input. In the next section we briefly describe several collision detection systems. Section 3 describes in greater detail a real-time simulation-based collision detection system based on octrees that is presently under development [2, 3]. When collision detection systems are implemented in software, questions of reliability must be raised. First, can they operate consistently and reliably in real time? In particular, simulation methods are likely to require time to update their world model and detect imminent collisions. Some methods described here claim to meet real-time requirements, perhaps for simplified test beds; ever faster computers will increase the possible complexity of simulations within fixed time constraints. Second, is the software error free? Simulations are likely to be more complicated than the software needed to control proximity sensors. Greater reliance on software presents a greater opportunity for software bugs.

Simulation approaches are also subject to problems resulting from inaccurate representation of the workspace. Even if an accurate representation can be created initially, maintaining accuracy after many updates to an object's position can lead to progressively greater degradation of the representation. Numeric instability issues for geometric models are discussed in [4]. For example, computing the rotation of an object using floating point calculation will result in an answer that is only a close approximation to the true rotation. The process of compounding many such operations makes these approximation errors grow ever larger. This problem can be avoided by collecting information about current object location directly from the environment whenever possible. Sensor-based methods are constantly acquiring information about the (local) environment. Simulation-based systems can receive joint angles directly from the robot arm before every position update/collision detection cycle. The simulation is therefore using the "true" rather than expected position of the moving arms at each time step. While the positions of the robot arm segments as computed from joint angles may not be completely accurate, their accuracy remains stable.

"Pure" simulation methods have some significant advantages over "pure" sensor methods. First, they require no additional hardware in most cases (although additional sensor information to monitor the state of the workspace is desirable, or may be required for a dynamic environment). Second, since simulation methods maintain a complete description of the workspace, sufficient information about the situation is available when an imminent collision is detected so as to give a range of possible responses (see Section 5).

2. A BRIEF SURVEY OF COLLISION DETECTION METHODS

Many researchers have studied path planning problems in both static and non-static environments (for overviews, see [5, 6]). Many collision detection algorithms have been developed to support path planners. With the expanded use of tele-operated robots in space, manufacturing, and the nuclear industry, the problem of real-time collision detection and prevention has become a significant problem in its own right.

One application of collision detection systems is to allow safe operation in a flexible assembly cell. Gouzenes [7] proposes an off-line method for computing the safe configurations for a group of robots. This approach requires no real-time collision avoidance system since the robots are never allowed into a configuration where collisions can occur.

Cameron [8] offers three different approaches to determining collisions based on geometric modeling of the environment. In the first, the motion is sampled at a finite number of times and interference detection between objects is performed at each time. In the second approach, models are created of the objects and their motions in space-time, and intersections between these 4-D entities are detected. In the third approach, models of the volumes swept out by the moving objects are created and checked for intersections.

Velocity and distance bounds are introduced in [9] as a means of detecting imminent collisions. Given two objects in space, we can determine the maximum velocity at which they are moving toward each other and the minimum distance between any two points on their surfaces. These bounds are used to determine the minimum amount of time dt that can elapse before these objects could possibly collide. The collision detection algorithm works by tracking this relation between each pair of solids in the world model. As the value for dt approaches zero for a pair of objects,

an imminent collision is reported. Simple enclosing solids are used as fast intersection checks to improve efficiency.

Minimum distance algorithms are discussed in [10] as a means of detecting imminent collisions in an off-line collision detection system with graphical simulation facilities. As objects move, a minimum distance measure is used to determine if any components of the pair of objects are about to collide. To reduce the number of pairwise comparisons between the components, each component was enclosed in a bounding box. The minimum distance algorithm (which is computationally expensive) was applied only to pairs of components whose bounding boxes were "close".

Yu and Khalil [11] present a collision detection system for a robot working in a fixed environment. The robot and the environment are modeled by means of simple primitives (i.e. spheres, cylinders, parallelepipeds, cones and planes). The authors observed that, in spite of the simple methods used for modeling, an "on-line" application based on testing the intersection of the robot links with all obstacles at each control point is not practical. In order to accelerate the calculation of the collision detection algorithm, a table look-up procedure is used. The representation of the free space is carried out by the discretization of the joint space and is stored in a table structure. This table is used to map the position of a robot link to the obstacles that lie close to that link. Thus, the number of intersection tests performed at each sample is reduced. This approach does not appear to be real-time.

The use of *region octrees* (see Section 3) to support collision detection is presented in [12], where an off-line system for testing plans generated by the planning system is proposed. To ensure that no collisions occur, the planned motions are sampled at discrete times and interobject interference is checked for. A region octree is constructed *a priori* for static objects in the environment. At each sample, a region octree is constructed for each moving object and static intersections tests are performed by parallel traversal of the octrees. A similar approach using region octrees to support off-line collision detection for plans is given in [13]. Region octrees have also been proposed to support path planning, both in three dimensions at discrete time slices [14] and in four dimensions (three spatial dimensions and time) [15]. Herman [16] suggests an approach based on comparing the region octree representation of the workspace against primitive shapes describing the swept volume described by robot arm motion, rather than against another region octree. This approach moves away from the region octree representation, and toward the octree representation described in Section 3.

These examples are but a few of the off-line methods for determining if robots in a predetermined environment will collide. Real-time collision detection methods are not so common, and until recently required special-purpose computing hardware.

A method using hardware-implemented clipping algorithms is presented in [17] to support real-time collision detection in a robot simulation program implemented on a graphics workstation. When a robot arm moves through the environment, the convex polyhedra which represent it is clipped against other objects in the geometric model of the robot's world. To facilitate real-time response, standard VLSI graphics hardware is used to perform the clipping tests between the robot and other objects. Smith describes how the graphics hardware could be integrated into a robot control system to provide on-line collision prediction and avoidance. Information about the robot's trajectory is provided to the collision detection system in advance. Information about the robot's surroundings is preprogrammed in this implementation. The detection algorithm is applied to points lying ahead of the robot on its current path, and imminent collisions are reported.

A real-time collision detection system directly incorporated into the control system of a robot manipulator is introduced in [18]. This system is implemented using a parallel machine with 64 microprocessors to manipulate a simulation of the workspace. The system is given the desired position of the gripper as input, and generates the appropriate sequence of joint angles to move the gripper to that position. Two types of imaginary force vectors—an attractive and a repulsive force vector—are generated. The attractive force vector is defined between the manipulator hand and the target position. This "pulls" the hand towards the target position. The repulsive force vector is defined between a part of the manipulator and an obstacle. This inhibits the manipulator from approaching the obstacle too closely. The joint angles at the next sampling time are

determined using these vectors and the world model, and are sent to the manipulator driver to perform the actual movements.

Freund [19] presents a real-time multi-robot non-linear controller and collision avoidance system for a pair of robots working on a conveyor belt with a common working space. This controller directly manipulates the equations for robot joint configurations, and is claimed to have some capability for determining collision avoidance strategies once a collision is detected. Freund's implementation requires only a single microprocessor.

Khatib [20] and Boissiere and Harrigan [21] each present real-time collision detection systems where the world model uses a *potential field* paradigm of repulsive forces around obstacles combined with attractive forces from the goal. Boissiere and Harrigan recognize the difference between a world model and a sensor-based local approach, though their implementation only supports the world model-based control.

All of the methods described above are primarily simulation, i.e. world model-based approaches. Few if any of these systems are concerned with how sensor data about the environment is integrated into the world model. In contrast, proximity sensor-based methods are primarily concerned with processing local information around the robot arm. A great deal of work has been done on the problem of *sensor fusion*, i.e. how to categorize and process the data acquired by sensors (see, for example [22] for access to this literature). Here we present samples of work related directly to collision detection (which requires only the recognition that an object has come too close to the robot, while not necessarily recognizing what that object is).

Sensor-based collision detection methods have used a number of different detectors. Infrared proximity sensors mounted on the gripper are proposed by Balek and Kelly [23]. In order to determine the distance to the obstacle, these sensors require in advance some information about the material composition of that obstacle.

A "skin" of tactile sensors was proposed by Vranish [24], with this concept extended to capaciflector proximity sensors for collision detection in [25]. A similar concept is described by Cheung and Lumelsky [1, 26], where several *sensor modules*, each composed of 16 pairs of sensors, are mounted on the robot arm. Each of these systems can detect objects at a certain range, say 10–15 cm. When an obstacle comes within range of one or more sensors, action is taken to avoid a collision. This typically means that the arm follows closely around the obstacle in the most straightforward path to the goal.

3. A REAL-TIME SIMULATION METHOD BASED ON OCTREES

We now briefly describe the use of a hierarchical data structure, the *N-Objects* octree, in a real-time collision detection safety system for multi-robot work environments. Further details on this work can be found in [2, 3]. Our initial intent when designing this system was to provide a real-time safety capability for multiple teleoperated robot arms in an (otherwise) static work environment. The safety system warns of imminent collisions between two robot arms, or between a robot arm and objects in the environment.

Our testbed (Fig. 1) is modeled on NASA's proposed space station operating bay containing two teleoperated robot arms and various objects to be manipulated and avoided by the arms. Each robot arm in our application has seven degrees of freedom, from seven joints, and each section of the arm is nearly cylindrical in shape. The operating paradigm is one of receiving an indication of movement by the arms in the form of a set of current joint angles, updating the representation of the environment to reflect that movement, and reporting any imminent collisions.

Each object in the working environment that can collide with another object is included in the world model. A constructive geometry approach is used to represent objects. Complex objects are described as the union of simpler primitive objects. For our representation, we have chosen *cylspheres*, cylinders with spheres on each end, to represent each link in the robot arms. The cylsphere both provides an acceptable approximation for the links and allows for efficient intersection tests. The robots' operating environment is not well modeled by cylspheres, so our representation also supports rectangular solids as additional primitives. The use of cylspheres and rectangular solids allows for a satisfactory representation of our working environment with only 35 primitives. We expect that the workspace for a wide range of applications can be modeled with

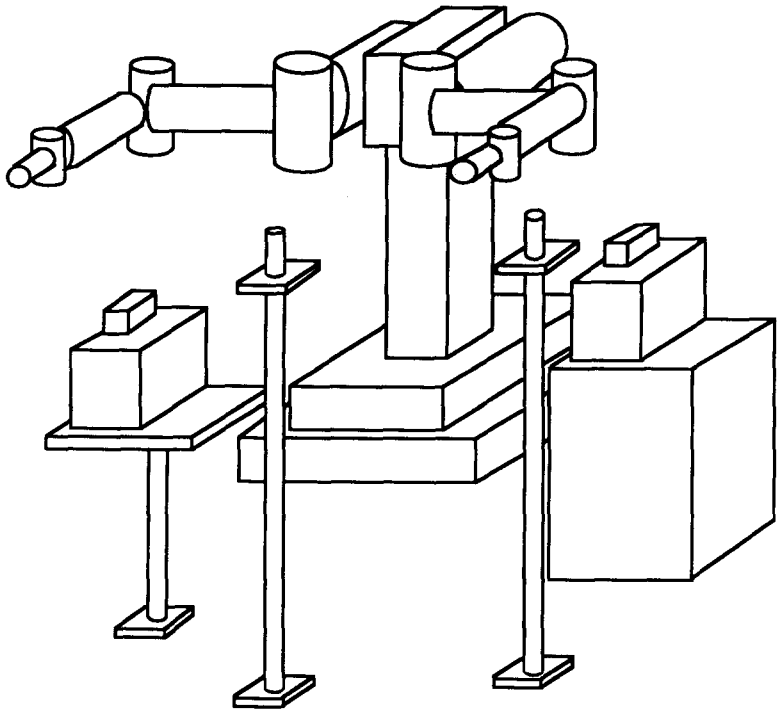

Fig. 1. Working environment used to test collision detection algorithms.

at most a couple of hundred primitive objects. New primitive shapes can easily be added to the system; all that is required are intersection procedures for the new primitive type with all existing primitive types.

To reduce the number of unnecessary intersection tests between primitive objects during the collision detection phase, an indexing scheme over the working environment is needed to determine which objects, and the primitives representing them, are close to each other and which are not. If we use no index, but simply compare all pairs of primitive objects, then the complexity of the collision detection algorithm will be $O(n^2)$ for n primitives. This level of performance is inadequate to provide real-time collision avoidance for any but the more trivial work environments.

We use an octree to maintain our spatial index. The octree is a hierarchical data structure that recursively subdivides a cubic volume into eight smaller curves (called octants) until a certain criterion, known as the decomposition rule, is met. This decomposition process is often represented as a tree of out-degree eight. Changing the decomposition rule gives rise to many varieties of octrees. The most commonly known form is the *region octree*. It is most appropriate for defining the shapes of homogeneous objects that are difficult to model with higher-level primitives. Beginning with a cube that encloses the set of objects to be modeled, splitting occurs until each leaf octant lies completely within an object or is completely empty (see Fig. 2). For an in-depth description of the octree and related data structures, see [27, 28].

Another type of octree, which we here refer to as the *N-Objects* octree, is widely used for ray tracing of images to simulate realistic lighting effects [29, 30]. The *N-Objects* octree subdivides space into octants, as does the region octree. However, the *N-Objects* octree stores a list of the objects that inhabit each node. Beginning with a cube that encloses all of the objects, splitting occurs until no more than N objects lie in any leaf node. The value we choose for N may depend on the characteristics of the environment being modeled and the task being performed, although we have found that for our application, running time is not sensitive to the value of N [2].

We chose the *N-Objects* octree to index the world model for our collision detection system for a number of reasons. The *N-Objects* octree provides a compact index describing the distribution of objects in the environment. An appropriate decomposition rule keeps the size of the tree small which, as we shall see, allows for fast updating. The *N-Objects* octree is a dynamic structure which easily adjusts to changes in the environment through the splitting and merging of octants. Equally

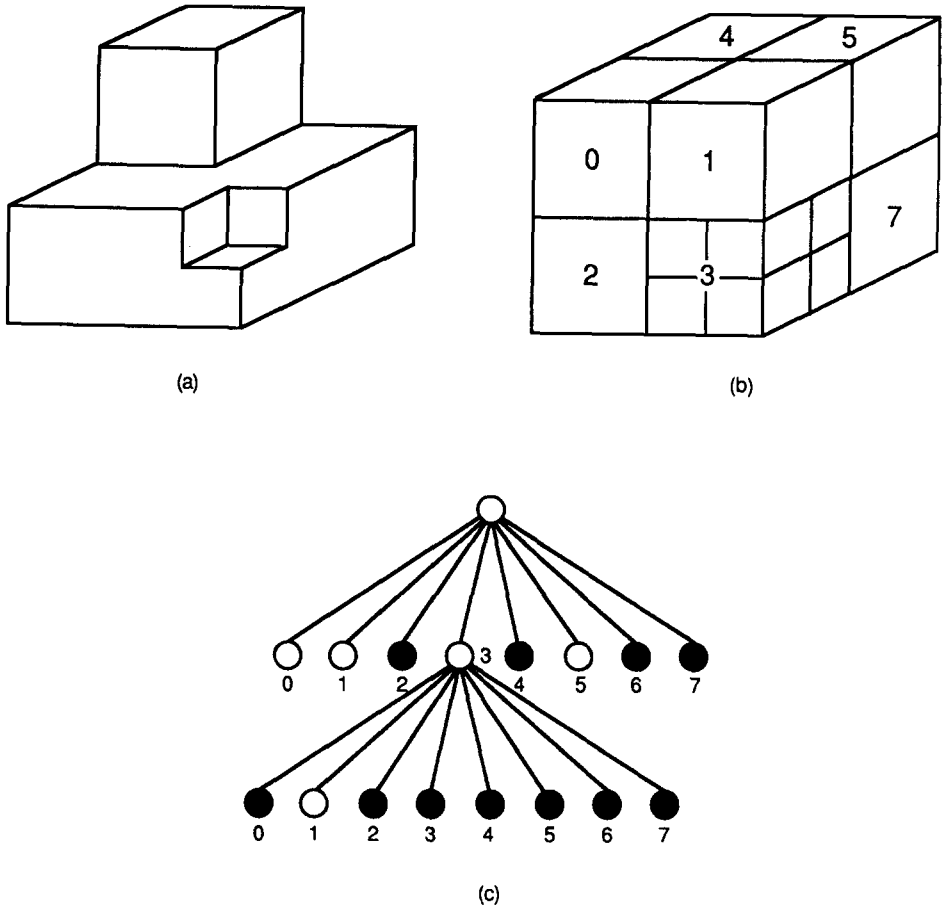

Fig. 2. An example region octree: (a) the object; (b) its region octree block decomposition; and (c) the resulting tree structure.

important to the efficiency of the system, small object motions rarely require that the structure of the tree be changed since objects rarely move from one octant to another. As a result, our system operates with an average cycle time of 30 ms (and under 60 ms in the worst case) for our test simulations.

We now turn to the problem of collision avoidance. We don't actually want to detect collisions. We wish to detect *imminent* collisions, with the intention of avoiding them. Thus, the collision detection system shall issue a warning when two objects come "too close" to one another. When a certain distance between objects is to be maintained, the standard technique for a static environment (such as in VLSI design) is to extend each primitive by 1/2 the minimum tolerance distance in all directions (which we refer to as the *safety buffer*). However, this is not a suitable approach when the safety buffer is to vary in size with an object's velocity. An alternative approach is to extend only moving objects by their variable width safety buffer. Now, whenever any primitives overlap, a collision warning can be issued.

The first step in modeling the world is to build the octree. We begin with an empty cube enclosing the working environment represented by the root of the octree. Objects (some of which have been extended to include their safety buffer) are added to the tree one at a time, and splitting is performed as directed by the decomposition rule. Figure 3 shows a 2-D workspace stored in an N-objects quadtree with $N = 5$.

After initial construction of the index, operation of the safety system is viewed as a series of time steps or *cycles*. During each cycle, the system receives two sets of joint angle values corresponding to the current configuration of the two robot arms. Using these values, simple kinematic transformations are applied to determine the current position of each primitive in Cartesian space. For each arm, we rotate each joint (and all joints dependent on it) to the positions specified by

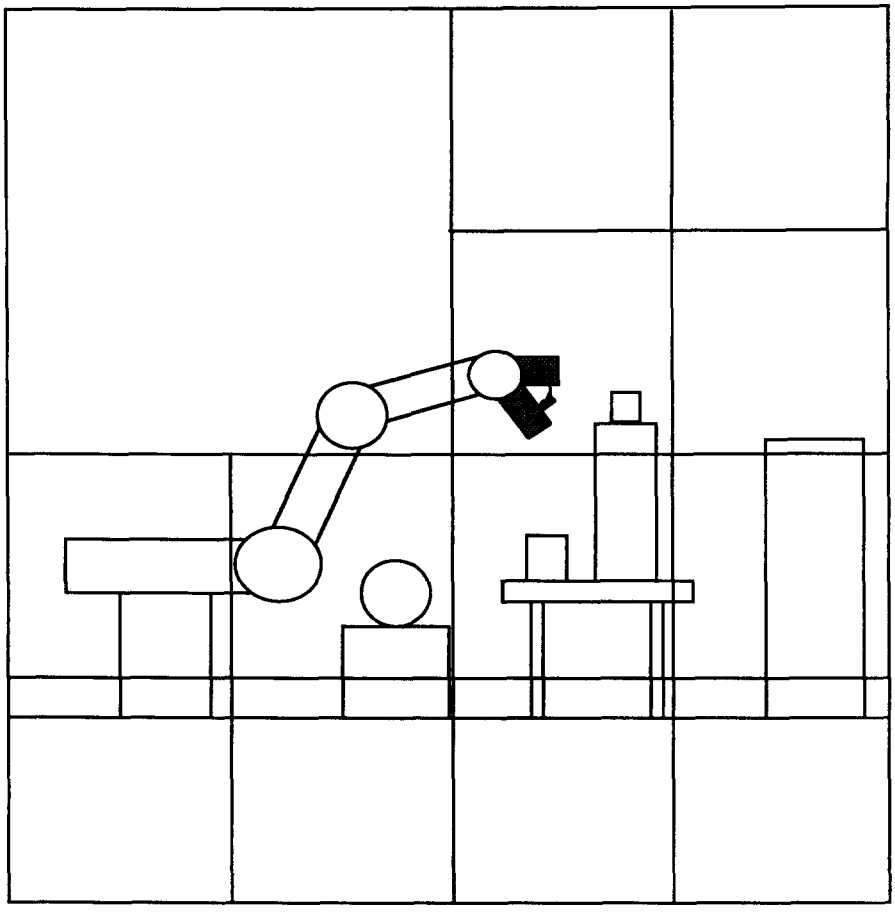

Fig. 3. Decomposition for sample environment using a five-objects quadtree.

its corresponding joint angle. As a byproduct of these transformations, the locations of objects currently attached to the end effectors of the robot arms are also updated.

The second step in the update process modifies the octree to reflect any changes in position of the robot arms. We observed that the tree structure changes only when an object exits an octree node (causing the object to be deleted from that node) or moves into a new mode (causing the object to be inserted). These events, in conjunction with the decomposition rule, may cause nodes in the tree to merge or split. An object can only enter nodes that are *neighbors* of a node in which it currently resides. Two nodes are considered neighbors if they share a face, edge or corner. The octree allows us to perform efficient updates by ignoring parts of the tree where no movement has occurred. Due to the small length of a time step, and the large node size relative to the distance traveled in one time step, it is extremely rare that an object actually enters or leaves a node. Collision checking must also be incorporated into the update procedure. After an object moves, all objects in all nodes that contain it are checked for possible collisions. In the octree, multiple moving objects may reside in the same node or a moving object and a static object may share more than one node. To eliminate any redundant intersection tests, we keep track of which pairs of objects have been checked for collisions during the current cycle. Thus, a complete (and relatively expensive) intersection test between a pair of objects will be performed at most once (although our update algorithm may check several times to see if a given pair has been tested).

Our collision detection algorithm was implemented using the C language under the UNIX operating system on an Intel 80386 CPU with math coprocessor running at 20 MHz. This configuration matches NASA's planned computing environment on the space station. Our testing consisted of running a series of simulated tasks using a fixed working environment (Fig. 1). The tasks ranged from 150 time steps to 2000 time steps in duration.

The average time per cycle (processing one complete set of joint angles and checking for collisions) over the three tasks was about 30 ms. However, the actual time for each cycle varied depending on how many primitives moved during that cycle. For example, all seven links of both arms moving required more computation time then if the end of one arm was moving. This is because the first case requires checking the octree 14 times for possible updates (since 14 objects have moved) whereas the second case requires checking the octree only once. The time required for each cycle of the algorithm was measured using the system clock, which had a resolution of 10 ms. The observed upper bound for the range of update cycle times was under 60 ms.

4. BUFFER ZONES AND VELOCITY

In a dynamic environment, several factors go into determining the minimum distance allowed between a robot arm and other objects. In general, sufficient space must be allowed so as to detect imminent collision, and then respond to it. Responding may mean stopping all motion of the arms, or it may mean deflecting their motion. For a sensor-based system, we might expect detection time to be extremely short once an object moves within range of a sensor. Stopping requires that the arm controller first receive the command to stop. It must then either change the direction of motion, or engage the brakes. Finally, the arm must actually stop or change direction, which may in turn cause oscillations or bending in the arm.

A simulation-based approach requires at least two steps before an imminent collision can be detected. First, the current position of the arm (possibly defined by the current joint angles) must be passed from the arm controller to the collision detection system. The simulated representation must then recognize that a collision is about to occur and issue a shutdown or evasive action command.

Cycle times for the controller to propagate joint positions to the detection system can vary widely between systems, ranging from only a few milliseconds (the expected time for the controller NASA intends to use on the space station) to 50 ms (as required for NASA's current test robots). The collision detection algorithm described in Section 3 requires 10–60 ms to issue the shutdown command; the system described in [19] has similar requirements. Time to actually stop depends primarily on the speed and mass of the arm. It also depends on whether stopping is effected by a change in acceleration by the arm motors (i.e. a change in direction of motion), or by throwing the brakes. Typical maximum speeds are around 24 in. s^{-1}; controlled stops from this speed require around 6 in. and nearly 1 s. Throwing the brake can stop the robot near instantaneously (perhaps 10–20 ms), but causes wear, possibly damage, and at the least disrupts calibration. Thus, a stop caused by controlled change of direction is greatly preferred. At a more typical operating speed of 6 $in.^{-1}$, this requires perhaps 2–4 in. and one-third of a second.

From the above analysis, we see that actual stopping time dominates the operation, and that current real-time simulations are fast enough so that computation time is only a negligible part of the process. We also see that controlled stops are important—which implies that knowledge of safe directions of motion can avoid emergency braking.

In a dynamic environment, a collision detection system should make provision for the relative speed of moving objects, since this is the dominating factor in controlled stopping time and distance. If an object is moving at less than maximum speed, the minimum safety buffer for that object can be reduced. However, we must also consider the accuracy provided by the world model in both shape and position of objects. For example, the cylsphere representation of Section 3 provides only an approximate model for the links of the arms, and in some cases is in error by more than 1 in. This is large compared to the required safety buffer size for typical operating speeds when we are trying to protect against rare, accidental collisions (i.e. we assumed instantaneous stopping by throwing on the brakes). Under these conditions, the overhead incurred by changing the model for the arms to account for varying speeds was felt unjustified for initial implementation of the system described in Section 3. Instead, we used a fixed tolerance for each arm link such that at least 1.2 in of buffer area beyond the approximated boundary of the robot arm is provided by the cylsphere. This is an acceptable approximation for our target environment since we did not expect that the operator will ever intend to have two arms (or objects) within less than 2 in. of each other (the exception being when the operator wishes to grasp an object—see Section 6). Future

implementations of our system allowing controlled deflection of motion rather than emergency-braked stops would benefit from safety buffers whose size is based on arm speed.

5. IMMINENT COLLISION RESPONSE

Once the collision detection portion of the safety system has determined that a collision is imminent, some response is required. The most obvious response is to stop all movement of the arms, and then either remove the offending object or reset the arms at their home position. This may not be desirable in a number of situations. If the arm is moving at relatively high speed, then sudden stopping of the moving arms is not good for the hardware, leading to shaking and stress on the entire robot and wear on the brakes. Alternatively, it may not be possible to stop the arms in time to avoid a collision. While this case might be avoided by maintaining a sufficiently large safety buffer, special situations such as two arms moving toward each other at maximum speed might require an unacceptably large safety buffer if room to stop safely must be guaranteed. Finally, if the arm is about to be hit by a moving object over which the safety system has no control, then the arm must take evasive action.

Simulation-based methods can direct the robot arm to make slight course adjustments to avoid collision since they can know the relative motions of the objects in question. A pure sensor-based approach may have accurate information about the position and velocity of objects within sensor range, but no information about objects beyond sensor range. Thus, a sensor-based system may not have sufficient information about which direction is safe to deflect the robot arm. Thus we see that collision response may require more information than is available from the sensors, and that some workspace simulation may be desirable even when sensor information is available.

6. GRIPPING

One task that is particularly difficult to integrate with a collision detection system is that of gripping an object. The problem is that this is a type of controlled collision—by its very nature it will be detected as a collision by any naive collision detection system. Thus, the system must be able to distinguish between intentional and unintentional collisions. When a robot gripper is about to collide with an object, the system should not report an imminent collision if the operator is actually attempting to pick up that object.

A simple solution to this problem would be to interrogate the operator to determine his intentions. However, as little interaction with the operator as possible is desired. An alternative approach might be to declare ahead of time all objects that will be manipulated by the robot to be compatible with the gripper (i.e. no collision response should be triggered by these objects coming together). Unfortunately, this still leaves open the possibility that the gripper may unintentionally collide with one object while trying to grasp another, or even of unintentional collision between the gripper and the object to be gripped at some future time. A third approach would be to adopt behavior heuristics. In particular, when the gripper approaches an object while moving at a low speed, assume that the operator is trying to manipulate that object.

A sensor-based collision system would be able to perform the gripping task provided that it maintains information on the velocity of objects within sensor range, and adjusts the safety buffer accordingly. With the simulation method, we can also reduce the collision buffer for reduced velocity as described in the previous section. It is reasonable to expect that as the arm comes closer to an object to be grasped, it should slow down. As the distance approaches zero, so will the velocity. A system based on velocity vs distance bounds, such as reported in [9] should be able to gracefully adapt to gripping operations.

The octree method of Section 3 provides an additional advantage that can be used during gripping operations. Its flexible means for indexing 3-D space easily supports dynamic modeling of robot arms. If we wish to change the model of the arm based on the type of task it is performing, we simply delete the model of the old arm from the tree and insert the new model. For example, when an arm is performing gross motions, for efficiency reasons we may wish for the entire gripper to be represented as a single primitive which completely encloses it. However, when the gripper is being used to grasp an object, a more detailed model is desired. This capability is supported by

deleting the coarse model and inserting the detailed model at the appropriate time. In this way we gain sufficient precision to support variable-width safety buffers for low-velocity gripper motion.

7. CONCLUSIONS

In this paper, we have described a number of collision detection systems, with our primary interest being real-time collision avoidance. These systems primarily fall into one of two distinct approaches: simulation (or world model) based systems vs sensor (or local information) based systems. World model systems typically do not update their world model based on sensor input (although they may update robot arm position based on joint angles received from the robot arm controller). Proximity sensor-based systems typically do not maintain a world model.

These two approaches to collision detection and avoidance actually complement one another, and should be integrated into a single system. A pure sensor-based method can supply a primitive level of safety, in that any object moving within a certain safety buffer can reliably shut down the system. A simulation-based system with its world model can be combined with higher level task and path planning modules, and can make more intelligent decisions when faced with an immenent collision.

The primary impediment to combining sensor-based and simulation-based systems is clearly the difficulty involved with integrating the sensor data into a world model. This process is often referred to as *sensor fusion.* As further progress is made in the area of sensor fusion, it will become feasible to collect data from the sensors, update the world model, detect immenent collisions and determine an appropriate avoidance strategy all in real time.

Acknowledgements—I wish to thank Greg Herb for his work in implementing the octree-based collision detection system described in Section 3, as well as his efforts in performing a large part of the literature search reported in Section 2. I also wish to thank Maureen Bartholemew and Chip Campbell of NASA/Goddard, first to Maureen for formulating the problem and supporting this work, and to both for their helpful conversations at critical stages of this project. This work was partially supported by NASA/Goddard SFS under Grant NAG 5-1183.

REFERENCES

1. E. Cheung and V. J. Lumelsky, Proximity sensing in robot manipulator motion planning: system and implementation issues. *IEEE Trans. Robotics Autom.* **5,** 740–751 (1989).
2. G. M. Herb, A real time robot collision avoidance safety system. Masters thesis, Virginia Tech, Blacksburg, VA (1990).
3. C. A. Shaffer and G. M. Herb, A real-time robot arm collision detection system. Submitted to *IEEE Trans. Robotics Autom.*; see also Computer Science TR 90–28, Virginia Tech, Blacksburg, VA (1990).
4. C. M. Hoffman, *Geometric and Solid Modeling: An Introduction* Morgan Kaufman, San Mateo, CA (1989).
5. M. Sharir, Algorithmic motion planning in robotics. *IEEE Comput.* **22,** 9–20 (1989).
6. S. H. Whitesides, Computational geometry and motion planning. In *Computational Geometry* (G. T. Toussaint, Ed.), pp. 377–427. North-Holland, Amsterdam (1985).
7. L. Gouzenes, Collision avoidance for robots in an experimental flexible assembly cell. *Proc. 1984 IEEE Int. Conf. on Robotics*, Atlanta, GA, pp. 474–476 (1984).
8. S. Cameron, A study of the clash detection problem in robotics. *Proc. 1985 IEEE Int. Conf. on Robotics and Automation*, St Louis, MO pp. 488–493 (15).
9. R. K. Culley and K. G. Kempf, a collision detection algorithm based on velocity and distance bounds. *Proc. 1986 IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, Vol. 2, pp. 1064–1069 (1986).
10. G. Hurteau and N. F. Stewart, Distance calculation for imminent collision indication in a robot system simulation. *Robotica* **6,** 47–51 (1988).
11. Z. Yu and W. Khalil, Table look up for collision detection and safe operation of robots *Theory of Robots*, Selected papers from *IFAC/IFIP/IMACS Symp.*, Vienna, Austria, pp. 343–347 (1986).
12. J. W. Roach and M. N. Boaz, Coordinating the motions of robot arms in a common workspace. *IEEE J. Robotics Autom.* **3,** 437–444 (1987).
13. V. Hayward, Fast collision detection scheme by recursive decomposition of a manipulator workspace. *Proc. 1986 IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, Vol. 2, pp. 1056–1063 (1986).
14. T.-H. Hong and M. Shneier, Describing a robot's workspace using a sequence of views from a moving camera. *IEEE Trans. Patt. Anal. Mach. Intell.* **7,** 721–726 (1985).
15. K. Fujimura and H. Samet, Path planning among moving obstacles using spatial indexing, *Proc. 1988 IEEE Int. Conf. on Robotics and Automation*, Philadelphia, PA, pp. 1662–1667 (1988).
16. M. Herman, Fast, three-dimensional, collision-free motion planning. *Proc. 1986 IEEE Int. Conf. on Robotics and Automation*, San Francisco, CA, pp. 1056–1063 (1986).
17. R. C. Smith, Fast robot collision detection using graphics hardware. *Proc. IFAC Symp. on Robotic Control*, Barcelona, Spain, pp. 277–282 (1985).
18. S. Kokaji, Collision-free control of a manipulator with a controller composed of sixty-four microprocessors. *IEEE Control Syst. Mag.* **6,** 9–14 (1986).

19. E. Freund, On the design of multi-robot systems. *Proc. 1984 IEEE Int. Conf. on Robotics*, Atlanta, GA, pp. 477–490 (1984).
20. O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots. *Proc. 1985 IEEE Int. Conf. on Robotics and Automation*, St Louis, MO, pp. 500–505 (1985).
21. P. T. Boissiere and R. W. Harrington, Telerobotic operation of conventional robot manipulators. *Proc. IEEE Int. Conf. on Robots and Automation*, Vol. 1, pp. 576–583 (1988).
22. A. K. Sood, M. Herman, M. M. Trivedi and H. Wechsler (Eds) Special issue on unmanned vehicle and intelligent robotic systems. *IEEE Trans. Syst. Man Cybernet.* **20** (1990).
23. D. J. Balek and R. B. Kelly, Using gripper mounted infrared proximity sensors for robot feedback and control *Proc. 1985 IEEE Int. Conf. on Robotics and Automation*, St Louis, MO, pp. 282–287 (1985).
24. J. Vranish, High performance conformal skin for robotics. Presented at the *1985 IEEE Int. Conf. on Robotics and Automation*, St Louis, MO (1985).
25. J. Vranish, R. McConnell and S. Mahalingam, "Capaciflector" collision avoidance sensors for robots. *Computers Elect. Engng* **17,** 173–179 (1991).
26. E. Cheung and V. J. Lumelsky, Towards safe real-time robot teleoperation: automatic whole sensitive arm collision avoidance frees the operator for global control. *Proc. IEEE Robotics and Automation Conf.* (1991).
27. H. Samet, *Applications of Spatial Data Structures; Computer Graphics, Image Processing, and GIS.* Addison-Wesley, Reading, MA (1989).
28. H. Samet, *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA (1990).
29. A. S. Glassner, *An Introduction to Ray Tracing.* Academic Press, San Diego, CA (1989).
30. J. D. MacDonald and K. S. Booth, Heuristics for ray tracing using space subdivision. *Graphics Interface* **89,** 152–163 (1989).

AUTHOR'S BIOGRAPHY

Clifford A. Shaffer—Clifford A. Shaffer received the Ph.D. degree in Computer Science from the University of Maryland at College Park in 1986, after which he worked for one year as a Research Associate at the Center for Automation Research, University of Maryland. Since 1987, he has been Assistant Professor in the Department of Computer Science at Virginia Tech. Dr Shaffer's primary research interests are in the area of spatial data structures. He has implemented several computer mapping systems based on hierarchical data structures, and is currently exploring the use of advanced data structures to support a real-time collision avoidance system for multiple robot arms. Dr Shaffer also has research interests in computer graphics (particularly in the area of geometric modeling) and computer aided education.