Proc. IEEE Int. Conf. on Robotics and Automation. Vol. 3. 1986. pp. 1044--1049

Fast Collision Detection Scheme by Recursive Decomposition of A Manipulator Workspace

Vincent Hayward *

Computer Vision and Robotics Laboratory
Department of Electrical Engineering
McGill University
Montréal, Québec, Canada

Abstract

This paper explains a simple method for fast collision detection in manipulator tasks. We show from examples taken in the literature that solutions to this problem can be chosen among a continuum of schemes, according to the method selected for representing the workspace and the robot, and the amount of computations performed before testing a particular trajectory. We then describe a method based on a recursive decomposition of the workspace, also referred to as an octree model, as a good tradeoff for a class of applications.

1. Introduction

The task of programming robots is widely recognized as a difficult activity, even in the case of the simplest applications. For this reason, research in robot programming has been evolving in two distinct directions. The first one is aimed at constructing entirely automatized robot programming systems such as described in [Lieberman77] or more recently in [Lozano85]. Another trend is to design systems that ease the work of a human programmer of robot applications by providing a set of programming aids such as graphics facilities, automatic reporting of performances, interfaces to powerful CAD/CAM systems, and pleasant user interfaces. These techniques, put together, lead to systems often called *off-line programming systems*. Such systems are described in [Paul83], [Wesley80], or [Ambler82]. The goal is to allow the design of robot applications without requiring their physical implementation. This has several positive consequences on the usefulness of robots. Fast prototyping of robot programs is made possible, provided the right set of simulation tools, and the integration of robotic applications within an existing or new manufacturing process is made easier.

One of the many reasons that can cause a robot program to fail is the collision of moving parts involved in the robot motion with objects located in the workspace. Among the tools that a robot off-line programming system should provide is a collision detector. Such a system takes as input a geometric description of a workspace and a robot trajectory (manipulator configuration as a function of time) and reports where and when a collision would occur, should the trajectory be executed.

* on leave from LIMSI, CNRS, France

2. Methods for Representing Solids

The central component of a collision detector is an interference detector. The design is based on a scheme for representing solid objects: the robot and the obstacles. We will take from [Rechiqua80] the terminology that we will use for discussing various possible schemes. We will review six schemes, attempting to order them according to how much of the original structure of a solid is captured in the representation, or in other terms, to increasing levels of enumeration, that is redundancy. Of course, any practical scheme will probably use a combination of two or several of these basic schemes. It is important to notice that conversions from more structural high-level schemes to more enumerative ones are easier to perform that the reverse conversions. For example see [Vossler85], or [Tosiyasu85].

Sweep Representations are the most structured schemes for representing solids. They directly rely on the fact that volumes can be generated by sweeping a surface. The most general representation consists of the description of a surface as function of the curvilinear abscissa and the description of the trajectory (position and orientation) of this surface. The generality of this scheme is great and not very well understood. However, elegant two dimensional algorithms based on similar schemes have been demonstrated [Brooks83a]. Three dimensional applications seem restricted to simple cases. This scheme is of primary interest for high-level descriptions of objects naturally decomposed in elongated elements elements [Agin76]. Also in a manufacturing context, they correspond to volumes generated by machine tools such as lathes or milling machines.

Constructive Solid Geometry are expressions of primitive portion of space and combinatorial (intersection, union, ...) and motional (translate, rotate) operators. These schemes also lead to varieties of possible choices for the primitive portions of space. The main varieties are bounded primitive solids or unbounded half-spaces. These schemes are extensively used in manufacturing systems [Wesley80].

Boundary Representations are the most familiar methods because they are mostly used in computer graphics. Solids are represented by their bounding faces, in turn represented by their bounding edges, in turn bounded by vertices. *Boundary Representations* can only represent polyhedra, other solids must then be approximated as polyhedra with many faces.

Cell Decomposition schemes are a three dimensional generalization of triangulations. A model consists of a collection of elementary solids, usually tetrahedra, meeting exactly at a common face, edge, or vertex. This scheme is generally useful to compute certain topological properties of represented solids. It is sometimes difficult to construct valid decomposition of arbitrary solids. There are extensively used in the context of finite element numerical methods.

Spatial Occupancy Enumeration schemes embody in a data structure the exhaustive **list** of *voxels* (volume elements), usually cubes lying on a square grid, that belong to the solid to be represented. These schemes has the advantage of the simplicity, but plainly applied lead to unrealistics amounts of storage in most of the cases. *Spatial Occupancy Enumeration* schemes are a special case of *Cell Decomposition* in which the cells are of identical size and shape.

Octree Representations are a special case of *Spatial Occupancy Enumeration* schemes. An octree is a hierarchical data-structure [Samet84] aiming at reducing the amount of redundancy inherent to the *Spatial Occupancy Enumeration* schemes. A cubic reference portion of three-dimensional space is divided into eight octants. Each octant can recursively divided into octants leading to a tree structure of order eight. Each node of the tree is labeled according to its position with respect to the solid to represent: exterior, interior, or recursively decomposed. Octrees have a number of properties and have led to many applications, including cartography, tomography, computer graphics, computer vision, robotics, computer aided design, etc....

3. Solid Modeling Schemes and Interference Detection

We will now review the kind of computation involved in computing the interference of solids using the six schemes described above.

Sweep Representations are particularly suited to the creation of objects and high-level structured representation, but the analytical geometry associated with them can be arbitrarily complex, rendering difficult the task of determining if a point is inside a body, let alone determining the intersection of two solids. If the initial model is available under the form of a *Sweep Representation* the solution to the problem of interference detection will usually involve the conversion of this type of representation to a more enumerative one. Very degenerated cases could be used, but then, the advantages of *Sweep Representations* cease to be clear.

Constructive Geometry Schemes are conceptually attractive representations for computing interferences since there are based on the very principle of having primitive objects interfering to create specific solids. However, the problem of interference computation, equivalent to the problem of connectivity determination, leads to non trivial and computationally expensive algorithms.

Boundary representations have been first used for the specific problem of interference detection, because interference criteria can be easily established using the relations of edges to faces [Boyse79]. As we will see below, computational problems can be alleviated in restricted cases.

Cell Decomposition schemes in their generality have no advantage over *Boundary Representations*, however, assuming that the cells are sorted according to some spatial order, can lead to interference algorithms similar to those used for *Boundary Representations*.

Spatial Occupancy Enumeration schemes are the representation of choice for interference problems, spatial addressing is easy and determining if a point is inside a solid is trivial.

Octree Encoding will provide a solution for the storage of representation models with a sufficient accuracy. Interference detection with octrees is trivial, because the cells are sorted in a double way: there are sorted by spatial occupancy (characterized by the path, or address of the cell in the tree) and by hierarchy of size (characterized by the depth). Octrees lead to naturally dichotomic algorithms growing with the *logarithm* of the resolution. Several methods for encoding octrees are available according to the properties that need to be emphasized.

4. The Computational Problem of Interference Detection

Using *Boundary Representations*, the basic method is introduced in [Boyse79]. It is shown that interference determination, in the general case, is quite computationally intensive because of the large number of possible cases. Boyse explains that collision checking of a moving polyhedral solid is a restricted case because it is only necessary to detect the change from a non-intersecting to an intersecting condition. The collision of two solids can only occur when an edge collides a face interior or a face boundary. Assuming that the trajectory of vertices can be analytically described, the two basic tests can be analytically characterized. If a robot and its environment are represented by polyhedra, if Re and Rf are the numbers of edges and faces of the robot, and Ee and Ef these numbers for the environment, an interference test costs : $ReEe + EeRf + EfRe$ basic tests. This leads to quite large figures in practical cases and they grow quickly with the complexity of the scene. In Boyse's paper, analytical tests are developed in the case of pure uniform translation or rotations. A recent further improvement of this test can be found in [Canny84], it concerns the case of combined uniform translation and rotation. This is far from the test required by the moving parts of a robot having revolute joints. The motions of robot links actuated by more than two revolute joint are highly non-linear and very difficult to characterize analytically. Piecewise approximations will remain very computationally expensive. So far, analytical methods suffer a lack of generality.

Now, we turn our attention to more enumerative schemes that lend themselves to fast intersection detection algorithms. We might think taking advantage of their efficiency to apply them along sampled trajectories. Unfortunately,

such an approach implies that a new representation is constructed for each sample of trajectory for all the moving parts. Considering the computations involved in such a construction, again we encounter a computational problem.

In [Ahuja80], a method based on planar projections of the three-dimensional scene is developed. It relies on the fact that intersecting polyhedra will always be projected as overlapping polygons. However, the reverse is not necessarily true and overlapping projected polygons can be generated by non intersecting polyhedra. It is a conservative method, but not practical for arbitrarily complex environments. The probability of false alarms can be decreased by augmenting the number of projections, but then again, a computational problem can be encountered.

5. Using the Joint Space of the Robot

These methods have been developed for the purpose of planning collision free robot trajectories. The basic method has been pioneered by Udupa [Udupa77] for planning collision free motions for the Sheinman arm. Taking advantage of the spherical kinematic family of the robot, Udupa shows how to relatively simply build representations of polyhedral obstacles in the 'joint space' ‡ of the robot. The joint space methods apply well when the manipulator is kinematically simple. In order to counteract the combinatorial explosion of a six dimensional joint space, reported practical algorithms always involve some sort of decomposition based on the kinematic properties of particular robots, as well as simplifications of the environment representations [Brooks83b, Luh84, Lozano81]. An overview and a discussion of these methods can be found in [Gouzène84]. In [Park83], is reported a method to build a joint space collision map for two robots sharing a workspace. Albus's Cerebelar Model Arithmetic Computer or CMAC algorithm [Albus81] is used as a data-compression device. Although, in theory, the dimension of the space in which to represent the map is twelve, only a four dimensional map is reported to have been tested. Finally, a practical algorithm concerning three joints of a anthropomorphic robot is detailed in [Faverjon84]. The joint space is regularly tessellated and an octree is used as the data compression device. However, adding one dimension to the transformed space (one extra degree of freedom) would render the method impractical.

To summarize, joint space methods suffer the problem of a combinatorial explosion in the general case, although when they are applicable, they lead to the fastest collision detection algorithms since a large part, if not all the required computations are performed ahead of time, through the process of mapping obstacle in joint space. However, changes occurring in the manipulator workspace will be difficult to reflect in the transformed representation.

6. A Fast Scheme

For the purpose of our application, we had to develop a scheme that would allow collision checking along a given trajectories applied to a robot moving in an arbitrarily complex environment. Furthermore, in the context of our application, it was not possible to ignore links of the robot, and there were, at least, four elongated limbs actuated by seven joints. The goal was to design an interactive collision detector, implementable on a small computer based, say on a 68000 microprocessor.

Because the environments in which the robot was to operate were potentially very complex, thus not decomposable into simpler entities such as walls or pillars, and because the system was not to depend on the kinematic family of the robot, any joint space method had to be discarded. Sschemes based on collision detection of moving polyhedra using *Boundary Representations* were a reasonable alternative, but were likely to lead to complex algorithms growing quickly with the complexity of the description. Furthermore, floating point operations were preferably avoided. Consequently, we turned our attention toward a purely enumerative scheme.

6.1 Octree Free Space Representation

The method relies on the representation of the free space by an octree. Because of the regularity, octrees are likely to provide the simplest and fastest algorithms among purely enumerative schemes. A survey of octree encoding techniques and applications can be found in [Meagher82].

Many octree conversion algorithms are available in the literature, see [Dyer80, Samet80, Tosiyasu85, Yau84], for example. Octrees are easily derived from other representation schemes. They can be constructed from sensory data as described in [Connolly84] using ray tracing methods, which is quite an interesting property in a robotic context. Conversely, octrees lend themselves to be displayed by similar methods [Doctor81].

6.2 Storing the Octree

The main disadvantage of this scheme is the amount of storage required for storing the representation. Among the available schemes for storing octrees, in the framework of our application, we must choose a scheme that conserves the advantage of pre-sorting the cells. There are three main possible schemes.

Nodes can be listed according to some determined tree traversal method, preorder, postorder, etc... [Knuth73]. In that case only 2 bits exactly can be allocated per node. These are the most compact representations of octrees, however, traversing the tree in any other order than the underlying order, as well as dynamically modifying the tree, is very inefficient. For these reason, we used this method for the external storage of the representations.

With pointer representations, the simplest solution is to allocate an array of eight pointers per node. This method of storage is the fastest for the present collision detection algorithm. However, the storage requirements might reveal to be a problem for small computers. There are various

‡ sometimes called, the 'configuration space', or the 'state space' of the robot.

ways for reducing the amount of redundancy inherent to a pointer scheme with the cost of some processing overhead. But since there is little hope to obtain a storage reduction better than by an order of magnitude, we will not detail them here. Another problem is that it is difficult to predict how much memory will be required.

Another possible scheme is to use a *pyramid* of three-dimensional arrays. Given a 2^n by 2^n by 2^n array, a *pyramid* is a sequence of arrays at half resolution of one to the next. If 2 bits are allocated per cell, some computations are saved over a 1 bit allocation scheme. The storage requirements in the latter cases are uniquely dependent on the resolution †. At an extreme, it is perfectly possible to represent the workspace of the robot by the simplest data-structure: a three-dimensional array of bits. But in that case, we loose the hierarchical nature of the representation.

6.3 Building the Octree

A set of functions have been written to allow the conversion from a *Boundary Representation* of the robot workspace to an octree. For experimentation, the simplest set of functions have been implemented. Primitive solids are build by a sweeping method. The most internal function adds a cell located in $\{x, y, z\}$ to an existing octree. Such a cell represents the smallest entity. Segments are built by cumulative sweeping of a cell. Parallelograms are derived from segments, and rhombohedrons from parallelograms. Adjacent cells are coalesced as the tree is built to prevent the storage to grow out of control. All these functions use recursive divisions by two of the entity to be built. Of course, the method outlined above are not very efficient because all the elementary cells inside the solids to be represented are visited, and the tree is built starting from the leaves. However, experience shows that the construction of a 10,000 cells tree is a matter of minutes using a 68000 cpu.

Alternative methods would require the use of a test to decide if a cube is contained inside the solid to represent. In that case, trees could be built in a optimal top-down fashion.

The set of functions to perform the Set operations (Intersection, Union, and Complement) on octrees have also been implemented and thus complex solids can be constructed from simpler ones.

6.4 Interference Detection

Once the representation of the free space (or of the obstacles, it is equivalent) is obtained, the task of determining if a given point belongs to free space (repectively an obstacle) is simple. Starting from the coordinates of the point, recursive divisions by two determine in which octant the point belongs and if this portion of space is labeled 'free', 'occupied' or 'subdivided' by simple inspection of the corresponding octree node.

† if r is the resolution, we have: $\sum_{n=1}^{\log_2 r} (2^n)^3$ = 292 Kbytes, with the 1 bit scheme and r = 128.

Transformations on octrees such as translations, rotations (even multiples of 90°), erosions, growths, etc... are computationally intensive because they involve re-arranging all the nodes of the tree. This is an important justification for special purpose hardware.

If we would represent the moving parts of the robot by an octree, it would require either to re-build a new octree at each trajectory sample, or apply transformations, none of these methods being practical. We have experimented two alternative methods:

(1) The robot is approximated by bounding volumes made of cylinders ended by hemispheres. For each of these volumes a new octree is built from the original obstacle representation, grown by the corresponding radii. Growing the obstacles (respectively shrinking the free space) involves visiting all the leaves cells of the tree and adding (respectively removing) all the cells within a given distance. This is also a rather lengthy operation, but done only once. The major drawback is that the system must maintain more than one octree in memory, but collision checks can then be performed at optimal speed. They consist of determining the interference of segments with an octree. Once again, this is done by dichotomic subdivision, starting with the ends, because they are more likely to interfere first. Given a trajectory, the positions of the segment ends are determined very simply using the classical robot kinematics [Paul81].

(2) If we cannot afford maintaining several octrees, we can make use of Boyse's idea as collisions occur only when boundaries intersect. Control points are allocated on the boundary surface of the robot, and their list kept on a joint per joint basis. Each trajectory sample, the position of all the control points is transformed in absolute coordinates and interference checked.

In either case, for the test to be valid, trajectories need to sampled in such a way that no part of the robot travels more that the resolution step. Since the trajectoies were provided in Cartesian space, the determinant of the Jacobian of the robot was used as an indication of the sample step to choose.

7. Experimental Results and Conclusion

The amount of nodes required for representing realistic environments is difficult to characterize. Furthermore, it depends on how the principle frame axes are assigned with respect to the scene. However, it has been shown that the amount of nodes is of the order of the surface of the objects. Also, the amount of nodes is not multiplied by eigth whenever the resolution is doubled. The example we used grew from aproximately 3000 cells for a 64 *by* 64 *by* 64 grid, to 10,000 cells for a 128 *by* 128 *by* 128 grid. This finding is in favor of a pointer-based implementation, over a pyramid scheme if high resolutions are needed. Of course, the

length of the algorithms hardly changed by doubling the resolution.

It has been found that, on the average, collision or not could be determined at half of the depth the tree. That made, on a processor such as the 68000 with fast memory, the cost for the test of a single point be 200 microseconds on the average, the internal loop being coded in a high level-language (C).

For method (1), considering that no more that 100 points are representing the stick model with the desired accuracy, and that only half of them need to be checked on the average, the check of sample of trajectory could be performed in 10 milliseconds for the complete robot. This figure is very little dependent on the complexity of the scene.

For method (2), considering that the three-dimensional geometrical transformation of a vector performed in fixed point arithmetic on the same processor costs approximately 300 microseconds, it is possible to perform one check per second of a robot bounded by 2000 points. This remains within acceptable limits for interactivity.

For a *pyramid* coding of the octrees, assuming a fast, multiplication-less addressing of a three-dimensional bit array, the reported times are equivalent.

The collision detector has been interfaced to the output of the RCCL Robot programming system [Haywardd84], allowing the user to check ahead robot programs. Interfaces to CAD systems need to be implemented as well as means for constructing the representations from sensory data.

8. References

[Agin76] Agin, G. J., Binford, T. O., "Computer Description of curved objects," *IEEE Trans. Computer*, Vol 25, No 4, April 1976, pp 439-449.

[Ahuja80] Ahuja, N. et Al., "Interference Detection and Collision Avoidance among Three Dimensional Objects," *First Ann. Nat. Conf. on Artificial Intelligence*, Stanford, 1980.

[Ahuja84] Ahuja, N., Nash, C., "Octree Representation of moving objects," *Computer Vision, Graphics, and Image Processing*, Vol 26, No 2, May 1984.

[Albus81] Albus, J., "Brains, Behavior, & Robotics," McGraw-Hill, 1981.

[Ambler82] Ambler, A. P., Popplestone, R. J., Kempf K. G., "An Experiment In the Offline Programming of Robots," *12th ISRR*, Paris, June 1982.

[Baer79] Baer, A., Eastman, C., Henrion, M., "Geometric Modeling: A Survey," *Computer Aided-Design*, Vol 11, No 5, September 1979.

[Boyse79] Boyse, J. W., "Interference Detection Among Solids and Surfaces," *Communications of the ACM*, Vol 22, No 1, 1979.

[Brooks83a] Brooks, R. A., "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol 13, No 3, April 1983.

[Brooks83b] Brooks, R. A., "Planning Collision-Free Motions for Pick-and-Place Operations," *International Journal of Robotics Research*, Vol 2, No 4, Winter 1983.

[Canny84] Canny, J., "Collision Detection for Moving Polyhedra," AI Memo No 806, MIT, October 1984.

[Connolly84] Connolly, C., "Cumulative Generation of Octree Model from Range Data," *IEEE First International Conference on Robotics*, Atlanta, June 1984.

[Doctor81] Doctor L. J., Torborg J. G., "Display Techniques for octree-encoded Objects," *IEEE Computer Graphics and Applications*, July 1981.

[Dyer80] Dyer, C. R., "Region Representation, Boundary Codes from Quadtree," *Communications of the ACM*, Vol 23, No 3, pp. 171-179, March 1980.

[Faverjon84] Faverjon, B., "Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator," *IEEE First Symposium on Robotics*, Atlanta, June 1984.

[Gouzène84] Gouzène, L., "Strategies for Solving Collision-free Trajectories Problems for Mobile and Manipulator Robots," *International Journal of Robotics Research*, Vol 3, No 4, Winter 1984.

[Hasegawa79] Hasegawa T., Hirochika I., "Modeling and Monitoring a Manipulator Environment," *Proceedings of the Sixth IJCAI*, Tokyo, August 1979, Vol 1.

[Hayward84] Hayward, V., Paul R. P., "Introduction to RCCL: A Robot Control "C" Library," *IEEE First International Conference on Robotics*, Atlanta, June 1984.

[Meagher82] Meagher, D., "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, Vol 19, No 2, June 1982.

[Knuth73] Knuth, D. E., "The Art of Computer Programming, Volume 1 / Fundamental Algorithms", Addison-Wesley, 1973.

[Lieberman77] Lieberman, L. I., Wesley, M. A., "AUTOPASS: an Automatic Programming System For Computer Controlled Mechanical Assembly," *IBM J. Res. and Develop.*, No 21, 1977.

[Lozano85] Lozano-Pèrez, T., "An Approach to Automatic Robot Programming", USA-France Robotics Workshop, University of Pennsylvania, November 1984.

[Lozano81] Lozano-Pèrez, T., "Automatic Planning of Manipulator Transfer Movements," *IEEE Transactions on Systems, Man and Cybernetics*, Vol SMC11, pp681-698, 1981.

[Lozano83] Lozano-Pèrez, T., "Spatial-Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, Vol C32, N0 2, February 1983.

[Luh84] Luh, J. Y. S., "A Scheme for Collision Avoidance with Minimum Distance Traveling for Industrial Robots," *Journal of Robotic Systems*, Vol 1, No 1, Spring 1984.

[Paul81] Paul, R. P., "Robot Manipulators: Mathematics, Programming and Control,", MIT Press, 1981.

[Paul83] Paul, R. P., "Sensors and the Off-Line Programming of Robots," Proceedings of Advanced Software in Robotics, AIM Publishers, Liège Belgium, May 1983.

[Park83] Park, W. T., "State-Space representation for Co-ordination of Multiple Manipulators", Proceedings of Advanced Software in Robotics, AIM Publishers, Liège Belgium, May 1983.

[Rechiqua80] Rechiqua, A. A. G., "Representations for Rigid Solids: Theory, Methods, and Systems", *ACM Computing Surveys*, Vol 12, No 4, December 1980.

[Samet80] Samet, H., "Region Representation, Quadtree from Boundary Codes," *Communications of the ACM*, Vol 23, No 3, pp. 163–170, March 1980.

[Samet84] Samet, H., "The Quadtree and related Hierarchical Data-Structures," *ACM Computing Surveys*, Vol 16, No 2, pp. 187–260, June 1984.

[Tosiyasu85] Tosiyasu, Kunii L., et al. "Generation of Topological Boundary Representations from Octree Encoding," *IEEE Computer Graphics and Applications*, Vol 5, No 3, March 1985.

[Udupa77] Udupa, S. M., "Collision Detection and Collision Avoidance in Computer Controlled Manipulators," *Proceedings of the Fifth IJCAI*, MIT, Cambridge Massachusetts, August 1977.

[Vossler85] Vossler, D. L., "Sweep-to-CSG Conversion Using Pattern Recognition Techniques,", *Computer Graphics and Applications*, Vol 5, No 8, August 1985.

[Wesley80] Wesley, M. A., "A Geometric Modeling System for Automated Mechanical Assembly," *IBM J. Res. Develop.*, Vol 24, No 1, January 1980.

[Yau84] Yau, M. M., "Generating Quadtrees of Cross Sections from Octrees," *Computer Vision Graphics and Image Processing*, 27, 211-238, 1984.