# A study of the clash detection problem in robotics

**Conference Paper** *in* Proceedings - IEEE International Conference on Robotics and Automation · April 1985

DOI: 10.1109/ROBOT.1985.1087245 · Source: IEEE Xplore

Stephen Cameron
University of Oxford

# A Study of the Clash Detection Problem in Robotics

Stephen Cameron
Department of Artificial Intelligence
University of Edinburgh
Edinburgh EH1 2QL, UK

**Abstract**

*To solve the clash detection problem we must decide whether a collision will occur between any pair of objects from a set of objects with known shapes and motions. We have considered three methods for performing clash detection: in the first we sample the motion at a finite number of times and perform interference detection at each time; in the second we create models of the shapes and their motions in space-time, and look for intersections between these four-dimensional entities; and in the third we create models of the volumes swept out by the objects. This paper is a brief, comparative study of these three methods, and includes some details of our experiments with the first two methods as implemented in a geometric modelling system.*

*This paper originally appeared in* Int. Conf. Robotics & Automation, *St. Louis, March 1985, pages 488–493. ©IEEE 1985. It was reformatted using* LaTeX *in September 1996. Contact address: Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, UK. Phone: +44 (1865) 273850. E-mail* `Stephen.Cameron@comlab.ox.ac.uk`

## 1 Introduction

Recently there has been a great deal of interest shown in the collision avoidance problem, that is, the problem of choosing paths for a set of objects so that they do not collide. In our research we have chosen to address the easier problem of clash detection, in which the paths are pre-ordained and we have to decide whether a collision will occur. (In the literature this problem is also referred to as the *collision detection* problem or the *interference detection* problem.) There are several reasons why clash detection is of interest in the context of robot programming and planning:

- The early generations of off-line robot programming systems are likely to be based on interactive graphics and mainly under human control. A fast, automatic clash detection facility can greatly assist the programmer who would otherwise have to spot a clash by visual means.
- Many planning problems concerning robot motions may be solvable by a simple, heuristic planner, which would require a clash detection facility to test its plans. Furthermore, even when a clash is found it may be possible to use information from the clash detection system to guide the formation of a new plan.
- Existing collision avoidance schemes are complicated and can handle only a limited range of shapes and motions.

In this research we have looked at methods for performing clash detection that work well with current geometric modelling technology, and should be extendible to handle a reasonable class of shapes and motions. We tried to produce efficient algorithms for solving clash detection, and implemented some of the methods in a geometric modelling system to get some feeling for their features and failings.

This paper gives an overview of the research, which is described in more detail in [4]. Section 2 deals with a theoretical framework which was used as a basis for the research, and section 3 describes the three methods considered. Sections 4–6 deal with each method in some detail, giving results where applicable, and section 7 concludes with some comparisons of the methods and the domains for which they may be useful.

## 2 Framework

It is often useful to consider an abstract formalisation of a problem so that we can isolate the practical difficulties from the theoretical difficulties. Suitable frameworks exist for the field of geometric modelling, and we have extended one of these frameworks to form a suitable basis for our work on the clash detection problem.

For geometric modelling the abstract framework is quite simple — instead of considering how a shape is modelled in terms of a series of bits in a computer memory, we regard a shape as being modelled by the *set* of points contained within the model. For example, a unit cube might be represented by the set

$$\{ (x, y, z) \mid 0 \leq x \leq 1,\ 0 \leq y \leq 1,\ 0 \leq z \leq 1 \}$$

For technical reasons restrictions are usually placed on the types of sets that are considered: for details, see [8].

This framework is suitable for static, three-dimensional shapes; to model shapes in motion we consider sets of points in a four-dimensional space, where the fourth dimension is temporal. Formally, if we model a shape at rest by the three-dimensional set $S$, and it has a motion which gives rise to a *location function* $\Lambda(t)$ at time $t$, then we model the shape and its motion by the set of points

$$\{ (\mathbf{x}, t) \mid \mathbf{x} \in \Lambda(t)(S) \}.$$

We use the name *extrusion* for the process of composing a three-dimensional set with a location function. It is now easy to state the clash detection problem formally — if we have two shapes, $S_1$ and $S_2$, which are modelled in motion by the (four-dimensional) sets $S_1^*$ and $S_2^*$, then they clash if and only if

$$\exists(\mathbf{x}, t) \text{ such that } (\mathbf{x}, t) \in S_1^* \text{ and } (\mathbf{x}, t) \in S_2^*$$

i.e., they clash if and only if

$$S_1^* \cap S_2^* \neq \emptyset.$$

## 3 Methods

In our study we considered 3 basic methods for performing clash detection; these are outlined below, and are dealt with in more detail in the following sections.

### Multiple Interference Detection

For this method we consider the shapes at a number of times in the time-span of interest, and see whether they interfere. As a very simple example, consider the two-dimensional situation shown in figure 1, where we have shown a square moving and clashing with a stationary triangle. If we test for a (static) interference between the objects at the situations shown then we will detect the clash. Performing interference detection is not too difficult — the problem is to choose the time-steps intelligently.

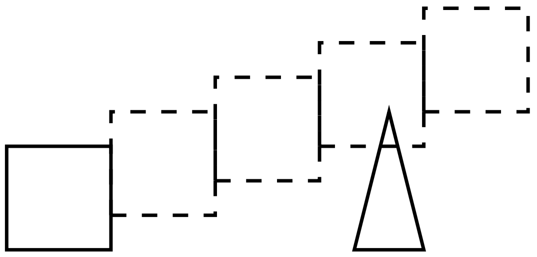
Figure 1:

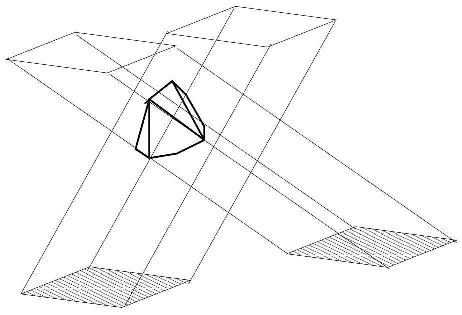
Figure 2:

### Four-Dimensional Intersection Detection

Here the emphasis is different; instead of testing for an interference in three dimensions, we work directly with representations of the four-dimensional sets that form the abstract models of the shapes in motion. Then to perform the clash detection we have to see whether the intersection of two (four-dimensional) sets is empty. Figure 2 shows an example in a lower dimension, in which two plates (shown hatched) are extruded to form intersecting wedges. The main difficulty with this method is to represent the four-dimensional sets inside the computer.

### Sweeping

The idea of the sweeping method is to take a three-dimensional object and generate a new three-dimensional object that contains the volume swept out by the object. Formally, if the object is modelled by the set of three-dimensional points $S$, and the location of the object is modelled by the function $\Lambda(t)$, then the sweep of the object is given by the set $Sw(S, \Lambda)$, where

$$Sw(S, \Lambda) = \{ \mathbf{x} \mid \mathbf{x} \in \Lambda(t)(S) \text{ for some } t \}.$$

Then if the objects collide their sweeps must intersect. Figure 3 shows a two-dimensional example, whereby two rectangles are swept to create intersecting envelopes.

Various forms of the first two methods have been implemented in a home-grown geometric modelling

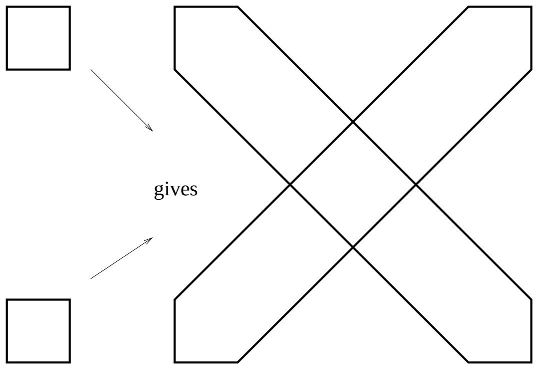

Figure 3:

system called ROBMOD. ROBMOD allows shapes to be described using a paradigm known as *constructive solid geometry* — that is, objects are described as set-combinations of other objects. ROBMOD can also be programmed in a simple, but useful, input language which includes features such as parameterised body descriptions, simple control flow, and a built-in, efficient interference detection function. An example of ROBMOD pictorial output is shown in figure 4, which shows a Unimation PUMA 600 robot and a simple workstation.

## 4 Multiple Interference Detection

To perform multiple interference detection we need to select a list of values of time for which we will test the objects for an interference. The simplest set of values we can choose are in an arithmetic progression, i.e. if we are testing for values of time in $[t_s, t_f]$ we can choose $\{ \tau_0, \tau_1, \ldots, \tau_n \}$, where

$$\tau_i = t_s + \frac{i(t_f - t_s)}{n}$$

This is the approach used in EMULA [7]. The problem still remains to choose $n$ — if $n$ is too small, we might miss the clash, and if $n$ is too large, we will waste time.

We have experimented with time-steps that can be varied by the program. ROBMOD has been given the ability to calculate the minimum distance between two objects; if we also give it an idea of the speeds of the objects, it can use quantities of the form

distance / speed

to guide its calculations of the time steps. Two algorithms have been implemented that use this form of cue. One I call "Achilles and the Tortoise", and the basic algorithm, `ClashAT()`, is sketched below. In `ClashAT()` we have a time variable, $t$, and we know that no clash occurs in the time interval $[t_s, t]$. The quantity

$$t_{safe} = \text{minimum distance / maximum speed}$$

gives a time interval in which a clash cannot occur, so if $t_{safe} > (t - t_f)$ we exit, and report "No clash". Otherwise t is incremented by $t_{safe}$, and we go around the loop again.

```
Boolean procedure ClashAT($t_s$, $t_f$, obj$_1$, obj$_2$)
  finished $\leftarrow$ false;      $t \leftarrow t_s$;
  while not finished and $t < t_f$
    begin  md $\leftarrow$ mindist(obj$_1$, obj$_2$, $t$);
    if md $\leq$ 0 then
      finished $\leftarrow$ true;
    else
      $t \leftarrow t +$ (md/maxspeed(obj$_1$, obj$_2$, $t$, $t_f$));
    end
  return ( $t < t_f$ );
endproc
```

The algorithm as shown suffers from a major disadvantage in that it will never detect a clash, as if the objects do clash it will infinitely recurse. This is easily avoided in practise, say by stopping the recursion when the minimum distance between a pair of objects is small.

```
Boolean procedure ClashDC( obj$_1$, obj$_2$, $t_0$, $t_2$)
  $s \leftarrow$ maxspeed( obj$_1$, obj$_2$, $t_0$, $t_2$);
  $d_0 \leftarrow$ mindist( obj$_1$, obj$_2$, $t_0$);
  $d_2 \leftarrow$ mindist( obj$_1$, obj$_2$, $t_2$);
  if $d_0 \leq 0$ or $d_2 \leq 0$      /* clash occurs */
    return true;
  elseif $s \times (t_2 - t_0) < (d_0 + d_2)$
    return false;
  else begin
    $t_1 \leftarrow (t_0 + t_2)/2$;
    return(  ClashDC( obj$_1$, obj$_2$, $t_0$, $t_1$) or
             ClashDC( obj$_1$, obj$_2$, $t_1$, $t_2$) );
  end
endproc
```

The second algorithm, `ClashDC()`, uses a "divide-and-conquer" strategy, and is shown above. `ClashDC()` works by considering both bounds of the time interval it is given. It first calculates the minimum distance between the objects at the extrema of the time interval, and works out whether the objects could possibly have approached, clashed and retreated within the time available. If not, the algorithm returns "No clash"; otherwise the interval is

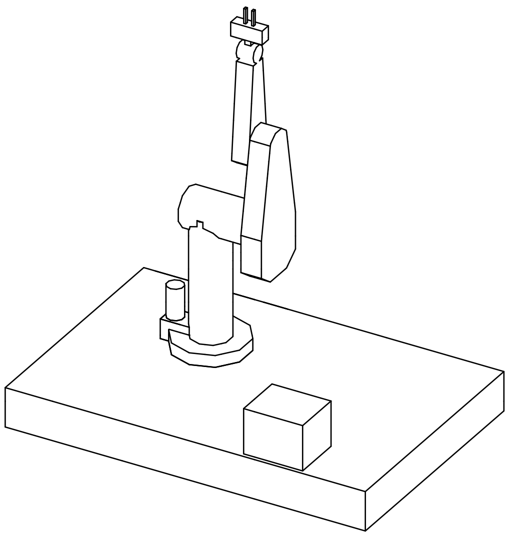
Figure 4:

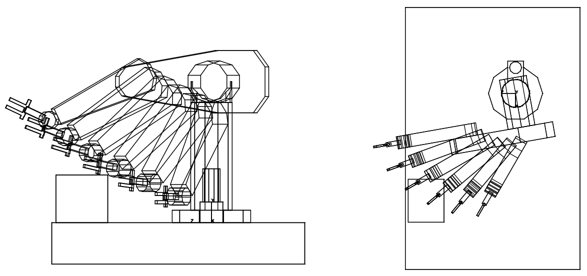
Figure 5:

split into two and the new intervals tested recursively. `ClashDC()` does always find a clash if it occurs, but it can do a lot of dumb division if the objects come close together for a long period of time.

**Example**

To illustrate these multiple interference detection algorithms I will use the PUMA workstation model from figure 4. Figure 5 shows two views of the robot going through a particular motion; each view shows the lower-arm assembly of the robot at times 0, 1, 2, 3 and 4, and also shows the entire robot and workstation at time 5. In fact four similar motions were given to the system, and figures figure 6a–d show snapshots of the lower-arm assembly at time 1.2 for each of the motions; I will refer to the four motions as motions (a), (b), (c) and (d). The motions are all formed by driving the waist, shoulder and elbow joints of the

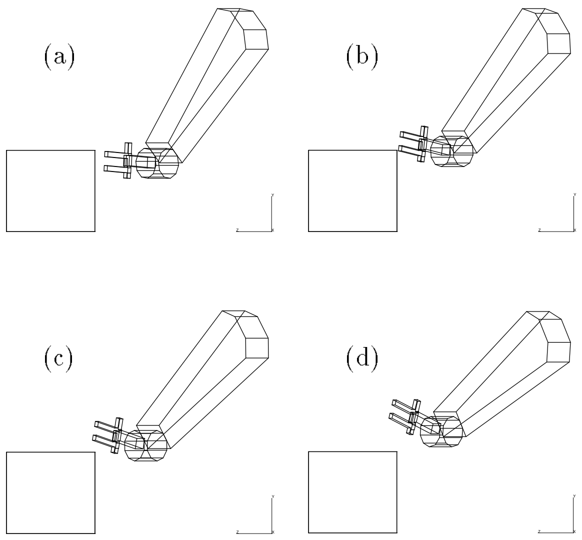

Figure 6:

robot at constant speeds. Motions (a) and (b) do result in a clash between the robot and the block, with motion (b) only just resulting in the two assemblies interpenetrating. Motions (c) and (d) do not result in clashes, with motion (c) giving a near-miss. All of these motions are considered over the time period [0,5], and the number of iterations for each of four multiple interference detection algorithms are tabulated below. `Clashlin()` is the simplest multiple interference detection algorithm described at the beginning of the section, which uses a constant time step, $\Delta t$. `Clashran()` is like `Clashlin()`, but for the time span of [0,5] chooses its times in the order 0, 5, 2.5, 1.25, 3.75, 0.625,.... $\Delta t$ in `Clashlin()` and `Clashran()` was chosen (arbitrarily) at 0.1, or 2% of the total time span, and `ClashAT()` was only expected to get within a small distance of a clash (approximately 0.1% of the total dimension of the robot). Note that even `Clashlin()` and `Clashran()` did happen to discover that there was a clash in motions (a) and (b).

| | Motion | | | |
|---|---|---|---|---|
| | (a) | (b) | (c) | (d) |
| `Clashlin()` | 12 | 12 | 51 | 51 |
| `Clashran()` | 3 | 41 | 65 | 65 |
| `ClashAT()` | 6 | 7 | 21 | 11 |
| `ClashDC()` | 3 | 9 | 15 | 7 |

These numbers are not particularly meaningful in themselves, but they do illustrate some useful points. Firstly, the number of iterations does peak for all the algorithms for the motions (b) and (c), as it is for these motions that it is difficult to decide whether or not a clash occurs. Secondly, all the algorithms do do better on the motions that result in a clash, namely (a) and (b), compared with motions (c) and (d). (This effect is particularly noticeable for `Clashlin()` and `ClashAT()` because the clashes happen to occur early in the motion.) Thirdly, "luck" can sometimes play a part; the number 3 for `Clashran()` and `ClashDC()` with motion (a) occurs because for this motion there happened to be a secondary clash occurring at time 2.5.

It should be noted that the time taken for ROBMOD to calculate the minimum distance between objects is significantly larger than the time taken to do interference detection; for the example given they are of the order of 5s and 1s respectively.

## 5 Four-Dimensional Intersection Detection

The main difficulty in implementing four-dimensional intersection detection comes in the representation of the four-dimensional sets. One way would be to represent the geometry and topological of the set in a manner similar to that employed by some solid modelling systems, but this is difficult to implement and maintain. We have chosen instead to use a constructive solid geometry representation, because it turns out that extrusion is distributive over the set operations; that is, the extrusion of the three-dimensional set represented by some boolean function of primitive shapes is represented by the same boolean function of the extrusions of the primitive shapes. Thus we only have to worry about extruding the primitives in our constructive solid geometry scheme. Mathematically even this operation is not difficult — if a primitive is expressed as a half-space function

$$S = \{\, \mathbf{x} \mid f(\mathbf{x}) \geq 0 \,\}$$

and has a location given by $\Lambda(t)$ then its extrusion, $S^*$, is given by the four-dimensional half-space

$$S^* = \{\, (\mathbf{x}, t) \mid f(\Lambda^{-1}(t)(\mathbf{x})) \geq 0 \,\}.$$

In practise even a simple half-space and motion can give rise to a complicated form for the extrusion. For example, if $S$ is a planar half-space with constant angular velocity then $S^*$ will generally be bounded by a helical hypersurface. We side-stepped this problem by considering only planar half-spaces going through piecewise-linear motions. Then the half-space

$$\mathbf{n} \cdot \mathbf{x} + d \geq 0$$

with location function

$$\mathbf{x} \rightarrow \mathbf{x} + \mathbf{c} + \mathbf{v}t$$

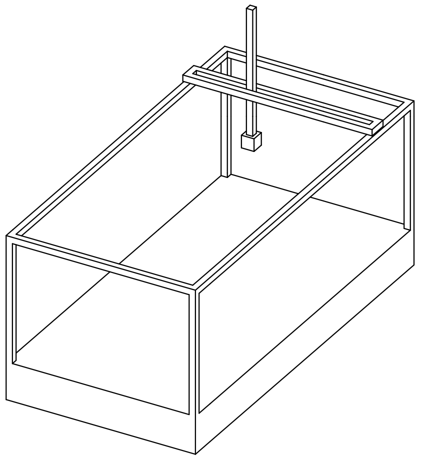
Figure 7:

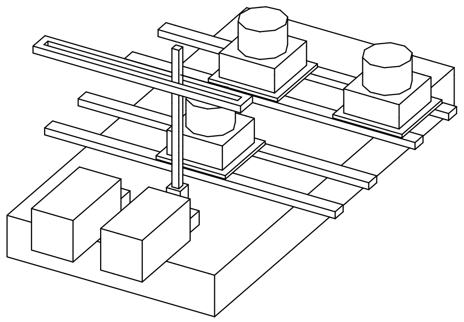
Figure 8:

gives rise to the four-dimensional half-space

$$\mathbf{n} \cdot \mathbf{x} - (\mathbf{n} \cdot \mathbf{v})t + (d - \mathbf{n} \cdot \mathbf{c}) \geq 0.$$

Intersection tests can be performed on these extrusions in a manner similar to that employed for three-dimensional constructive solid geometry descriptions. A method for performing intersection tests on half-spaces bounded by a larger class of hypersurfaces is described in [6]; such techniques may make four-dimensional intersection detection practical for other motion classes, but we have not followed up this train of thought.

### Example

A cartesian robot was invented as a suitable test for the algorithm. The basic robot is shown in figure 7; it consists of an arm which can be moved up and down in a trolley, and the trolley can move in a horizontal plane above the table. Figure 8 shows the workstation which contains three loaded pallets running on tracks, and a couple of feeders fixed at one end of the table; the trolley mechanism has been ignored for clarity. The test scenario was of the robot arm being required to make a point-to-point motion between its rest position (as in figure 7) and its final position above a feeder (as in figure 8). The arm is then in danger of clashing with the loaded pallet in the middle of the table. ROBMOD allows the motions of the arm and the pallet to be altered interactively and allows pictures of the situation to be drawn at any simulated time, and so the user is able to form his own opinion as to whether a clash occurs, or he can ask ROBMOD itself. Various motions were given to these two assemblies and the clash detection procedure solved all the problems that it was given correctly. Indeed the function has often proved itself to be more capable of testing for a clash than the author.

The implementation of this algorithm is still too young for us to say much about its speed with any confidence. However, the test results have been quite encouraging, as have the results of a statistical analysis in which we tried to elicit bounds for the expected time complexity of the algorithm.

## 6 Sweeping

The use of three-dimensional swept volumes to perform clash detection is quite intuitive, and has been pursued by several workers in the field. However, in the general case the main difficulty with the method is that it is very difficult to represent the swept volume explicitly. This problem is normally side-stepped in one of two ways. Firstly, the class of possible objects and motions may be restricted to ensure that the swept volume can be created. An example of this approach is seen in the LARS system [5], which allows spheres to be swept along straight lines or along circular arcs, thus forming cylinders or toroids respectively. The second way of side-stepping the problem is to only create an implicit representation of the swept volume. For example, Boyse [2] describes a system which performs clash detection for planar-facetted surfaces by sweeping the individual vertices, edges and faces separately.

The advantage of this method over explicit sweeping is that we do not have to recombine the sweeps of the individual features to form a coherent three-dimensional model. However, for complex geometry or motions we still have the problem of sweeping the various features. It should be noted that sweeping is equivalent to the extrusion operation followed by a projection of the extrusion, and this projection suppresses the temporal information in the data-structure. Thus it seems that sweeping is more difficult in the general case than extrusion. In practise there may be special cases that can be handled easily by sweeping (as in LARS) that make clash detection by sweeping preferably to four-dimensional intersection detection, but for the more general case we suspect that four-dimensional intersection detection will prove to have the advantage.

## 7 Conclusions

We have discussed here three different methods for performing clash detection, each of which has its own merits and failings. Multiple interference detection has the advantage of simplicity, and it also seems well suited to interactive environments as it can account for its actions by producing pictures of the situations that it is testing. There is also the advantage that we do not need to have an internal representation of the motions of the objects, but can make do with just sampling the locations of the objects. However multiple interference detection cannot easily handle some interesting cases, such as sliding contacts.

Four-dimensional intersection detection is elegant, and can handle "special" cases, like the case of a sliding contact, However, there can be difficulties in representing the four-dimensional sets when we have anything other than simple objects and motions. There would seem to be two ways out of this dilemma. One would be to treat the surface of the three-dimensional objects as facetted, and their motions as piecewise-linear, thus producing simple hypersurfaces in the model of the extrusion, and the second method would be to deal with a wider class of hypersurfaces directly. Which approach is preferable is still an open question.

Explicit sweeping is a useful method when it can be employed, as we then have a visual check available in the form of the swept volume. However it is not practical to extract the time at which a clash first occurs from an explicit swept volume. Implicit sweeping has a larger performance envelope, but both sweeping methods must be used with care if more than one of the objects is moving, as it is then possible for a pair of swept volumes to intersect without the objects having clashed.

We do not believe that any one of the methods discussed here is "best", as each can be useful in different applications. Furthermore, we can conceive of a composite algorithm that would first employ multiple interference detection when the objects are far apart, and then use one of the other two methods when a pair of objects are close together. This approach would have two advantages: firstly, multiple interference detection is then being used when it is most efficient; and secondly, we need only employ four-dimensional intersection detection (say) for some subset of the geometry in the database, and small intervals of time.

## Acknowledgement

This work was supported by a Science and Engineering Research Council studentship.

## References

[1] A. Baer, C. Eastman, and M. Henrion. Geometric modelling—a survey. *Computer Aided Design*, 11(5):253–272, September 1979.

[2] J. W. Boyse. Interference detection among solids and surfaces. *Communications of the ACM*, 22(1):3–9, 1979.

[3] R. A. Brooks and T. Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. In *IJCAI '83*, 1983. Also as MIT AIM-684.

[4] S. A. Cameron. *Modelling Solids in Motion*. PhD thesis, University of Edinburgh, 1984. Available from the Department of Artificial Intelligence.

[5] A. de Pennington, M. S. Bloor, and M. A. Balila. Geometric modelling: A contribution towards intelligent robotics. In *13th. Int. Symp. Industrial Robotics*, Chicago, 1983.

[6] D. M. Esterling and J. Van Rosendale. An intersection algorithm for moving parts. In *Proc. NASA Symposium on Computer-Aided Geometric Modeling*, pages 119–123, Hampton (VA), April 1983. Conf. publ. 2272.

[7] Jeanine Meyer. An emulation system for programmable sensory robots. *IBM J. Res. Dev.*, 25(6):955–962, November 1981.

[8] A. A. G. Requicha. Mathematical models of rigid solid objects. Technical Report PAP TM-28, University of Rochester, November 1977.

[9] A. A. G. Requicha and H. B. Voelcker. Solid modeling: A historical summary and contemporary assessment. *IEEE Comp. Graphics & Applications*, 2(2):9–24, March 1982.

[10] R. B. Tilove. A null-object detection algorithm for constructive solid geometry. *Communications of the ACM*, 27(7):684–693, July 1984.

[11] S. Udupa. Collision detection and avoidance in computer controlled manipulators. In *Int. Joint Conf. Art. Int.*, Boston, 1977.