NASA Contractor Report 175319

A Strategy Planner for NASA Robotics Applications

S. Brodd

Science Applications Research
Lanham, Maryland

Prepared for
National Aeronautics and Space Administration
under Contract NAS5-28200

National Aeronautics
and Space Administration

Goddard Space Flight Center

1985

NASA Contractor Report 175319

A Strategy Planner for NASA Robotics Applications

S. Brodd

CONTRACT NAS5-28200
JULY 1985

NASA

Acknowledgements

The work described in this report was performed for Science Applications Research under contract NAS5-28200 to NASA Goddard Space Flight Center. The assistance of Goddard employees Timothy Premack and Lloyd R. Purves is gratefully acknowledged. In addition, the inspiration of Edwin P. Cutler, former SAR employee, is recalled with appreciation.

Table of Contents

A Strategy Planner for NASA Robotics Applications

Introduction

A strategy planner is under development at Goddard Space Flight Center to automatically produce robot plans for the assembly, disassembly, or repair of spacecraft. The input to the planner is a geometric description of the assembly produced by a computer-aided design (CAD) system. Using this description, together with information about the capabilities of the robot to be used and its own knowledge of the conditions and restraints of the problem, the strategy planner executes a search through the field of possible manipulations of the parts of the spacecraft for each part's installation or removal, as required. Individual sequences are synthesized into an overall plan for the requested operation on the spacecraft.

Important features of the strategy planner include the use of a tree-structured knowledge representation for space-filling, geometric parts, the use of several heuristics based on given and inferred geometric data to limit the planner's search procedure, and the use of several generic algorithms to manipulate and extend, by adding new logical information, the geometric description of the spacecraft under consideration.

The entirety of the strategy planner is written in the Prolog language, which has proven to be an efficient and well-structured tool for this type of application. (Clocksin and Mellish, 1982). Knowledge from the several disciplines of mechanical engineering, mathematical geometry, and artificial intelligence has been synthesized in the design and implementation of the strategy planner.

Background and Related Work

Automatic plan generation (and subsequent execution) for real-world robotics is a topic of considerable current research, and has traditionally been of interest to the artificial intelligence community. Typically, industrial robots must be programmed by a human for each task they perform, be it welding automobile frames or painting refrigerators. Minor modifications in task descriptions can require major modifications in programming. Further, for precise manufacturing tasks, program specifications may be complicated and difficult to determine. Automatic planning seeks to generate these programs by computer, and so to employ the advantages of speed and accuracy when faced with new tasks, as well as to free human labor for design and creative use.

The goal of automatic planning in artificial intelligence is the generation of a sequence of instructions that will transform the initial state or set of initial states of some well-defined environment into a desired final state (or any one of a set of final states). The set of allowable instructions at any given point is

restricted by whatever conditions or constraints have been placed on the problem and by the particular configuration of the environment at that point.

Historically, this type of planning has dealt with blocks world kinds of applications, in which the various objects in an environment and their relationships to each other could be described by English words or sentences. (Nilsson, 1980). We are seeking to augment this body of work by greatly increasing the complexity of both objects and relationships, and so to be able to operate on real NASA hardware.

A considerable amount of research related to the area of automated strategy planning has been published. An overview touching on the topics of world modeling, task specification, symbolic spatial relationships, and grasp planning may be found in Robot Motion (Brady, et al., eds., 1983). The problem of knowledge representation schemes for solid objects has been addressed by Requicha (1980), and Lozano-Perez (1979, 1983) has written on the problem of planning motions through obstructed space. Boyse (1979) has investigated interference detection in a geometric modeling environment. An interesting system similar to the present project but with its principal emphasis on graphics simulation is described in Developments in Robotics 1983 (Bonney, M.C. et al., 1983). The strategy planner is part of and an extension to the work previously published by Premack, et al. (1984).

Strategy Planner

The design of the strategy planner incorporates two basic areas: a knowledge representation for the data -- geometric descriptions of space-filling, three-dimensional parts of spacecraft; and a software system to perform the plan generation.

Knowledge Representation

Key to the success of the strategy planner is the method of storing the large amount of geometrical, spatial, and logical knowledge available about the spacecraft under consideration.

Context-Free Grammar Representation

Many schemes have been developed for representing and storing information about mechanical parts, including detailed drawings with annotations of dimension, material, and the like to the databases of various computer-aided design systems, with their capabilities for graphics display and calculation of physical characteristics.

We have developed a knowledge representation that has proven to have a number of advantages for the strategy planner. The representation defines three-dimensional, space-filling objects as a set of primitive shapes combined in a tree structure. The primitives are the simple building blocks with which complicated parts can be constructed, and the tree structure defines the method of construction.

One basic type of primitive solid is represented, that of an arbitrary planar area swept along a path. From this general representation, we have chosen two subcases: solids of linear extrusion (or "lexsolids"), and solids of revolution (or "revsolids"). A lexsolid is defined by a two-dimensional curve and a three-dimensional vector. The curve, which must be closed, bounded, planar, convex and non-self-intersecting, is moved from its original position along the direction of and for the extent specified by the vector. The result is a space-filling shape; cylinders and cones are examples. See Figure 1 for an example of a lexsolid.

Similarly, a revsolid is defined by a curve, a line, and two angles. The curve, which in addition to the above restrictions must be coplanar with the line, is revolved about the axis defined by the line from the first angle (the starting position) to the second angle (the ending position). Again, a space-filling shape is defined; cones and toruses are examples. (Note that a cylinder could be defined as either a lexsolid or a revsolid). See Figure 2 for an example of a revsolid.

We allow two types of two-dimensional curves in defining these primitives: polygons composed of an arbitrary number of straight lines; and conic arcs (of which circles are the most common special case).

Polygons are defined as a list of coplanar lines, and each line is defined by two three-dimensional vectors, the first its starting point and the second its ending point. The lines are arranged in order so that the starting point of one line is coincidental with the ending point of the preceding.

Conic arcs are defined by a type, which is one of ellipse, parabola, or hyperbola, and six coefficients which define the shape of the curve. The location of the curve is defined by a vector called the "locus". Since circles are so commonly used, a special form has been adapted that gives the center and radius of the circle directly.

The shapes defined by the primitives are combined into the required part through the use of Boolean operators in a binary tree. At each level of the tree, two "subparts" are related by the operations of union, intersection, or difference. In the first two cases, either or both subparts may themselves be complicated objects defined by trees. The union comprises all points in either subpart or common to both; the intersection comprises only those points common to both subparts. In the case of difference, the first subpart may be a tree but the second must be a primitive within the first. The defined object consists of all points in the first subpart except those within the second. See Figure 3 for examples of union and difference operations.

The entire knowledge representation can be listed in the form of a context-free grammar. The productions for this grammar are given below. Terminal symbols are underlined; capital letters represent angular dimensions in radians and small letters represent real-valued dimensions. A complete "partlist" designates all those parts in a single assembly.

partlist --> part,partlist; part

part --> boolean(part,part); difference(part,subpart);
subpart

subpart --> lexsolid(curve,x,y,z); revsolid(line,curve,A,B)

curve --> circarc(center(x,y,z),radius(r));
conicarc(type(type),coefficients(a,b,c,d,e,f),
locus(x,y,z));
polygon(linelist)

linelist --> line,linelist; line

line --> line(start(a,b,c),end(d,e,f))

boolean --> union; intersection

type --> ellipse; parabola; hyperbola

[A,B] --> {real angles in radians}

[a,b,c,d,e,f,x,y,z] --> [real dimensions]

Figure 4 shows some examples from the Prolog implementation of the above context-free grammar.

This form of knowledge representation allows the specification of a large universe of complex geometries from a few simple primitives. The bulk of the information is logical rather than mathematical, and this means that executions of computationally time-consuming mathematical algorithms may be kept to a minimum in favor of faster logical algorithms. In addition, these Prolog data structure definition formats lend themselves well to the type of recursive search we use to perform strategy planning, and provide a database that, in contrast to that of many CAD systems, is easily readable and editable by a human.

Initial Graphics Exchange Specification

The input to the strategy planner may in general come from any computer-aided design (CAD) system; the particular input we have chosen to use is in the IGES format. The Initial Graphics Exchange Specification is a widely used digital representation for communication of product definition data. (Smith, 1983). The intent of the representation is to provide a uniform means for transforming data directly from one CAD system to another, without having to manually reenter information from blueprints and other printed documentation. The original IGES representation has been extended to

incorporate additional means for describing solid shapes in the IGES Experimental Solids Proposal. (Smith, 1984).

Both the original representation and the solids specification extension contain a large number of "entities" or informational constructs, providing for geometrical constructs capable of describing virtually any three-dimensional design, as well as such things as annotation data, dimensions, leader lines, and various sorts of relations.

We have found it useful to choose a subset of these many entities and to require that each IGES database input to the strategy planner contain only these entities. This subset of entities conveys purely geometric information, and corresponds closely to parts of our knowledge representation. Transmission of special information about particular parts of a spacecraft is also possible.

Specifically, the entities are: circular arc, composite curve, conic arc, line, transformation matrix, Boolean, solid of linear extrusion, and solid of revolution.

With the use of the IGES representation as the input medium to the strategy planner, we have gained the advantage of access to the databases of many CAD systems, at the relatively small cost of translating the IGES database into a format more immediately well-suited to the task at hand. See Appendix B for an example of the IGES definition format.

Software

The software system in the strategy planner comprises algorithms which perform the following operations:

- ingest the CAD geometric descriptions and translate these descriptions into the new knowledge representation
- execute the search procedure to find the disassembly or assembly sequence
- limit the search with the use of heuristics
- determine the feasibility of suggested operations on the database of part descriptions
- manipulate this database as the search progresses

Two examples of strategy planner Prolog code are given in Appendix D. In addition, see Appendix A for an example of the software's operation.

Ingest

The first part of the strategy planner software translates the incoming IGES database description of the assembly into the context-free grammar construction that will be used by subsequent procedures.

The code to "ingest" the IGES data is fairly complex, as a number of conceptual changes must be made. As seen by the strategy planner, a subject spacecraft exists in a single Cartesian frame of reference, with each of its constituent parts defined by solid primitives connected by Boolean operators as described above. All parts are explicitly described, so that the database is straightforward, and operations on it as a whole or in part are simple to implement and record. In IGES, on the other hand, the requirements of ease in entry prevail, so that each solid primitive and each defining planar curve are in general located within their own frames of reference. Connections between them are recorded in transformation matrices, describing the manipulations required to locate each entity within the overall design. Once defined, entities may be called repeatedly and situated with different transformation matrices. Also, Boolean connectives are defined in reverse Polish notation and are not integrated into the descriptions.

The operations of the ingest software are:

1. to translate each curve and solid primitive from the IGES representation to the context-free grammar representation

2. to explicitly define each primitive and complex part

3. to apply all the (possibly accumulated) scalings, rotations, and translations contained in the IGES transformation matrices to locate the entire spacecraft in one frame of reference

4. tc incorporate the Boolean logical data in the intrinsic fashion of the context-free grammar representation

See Appendix B for an example IGES to context-free grammar transformation.

Once the IGES data has been ingested, the strategy planner may begin its search procedure.

Search Procedure

Given the completely specified description of the assembled spacecraft, the strategy planner has the task of producing a sequence of robot instructions for assembling, disassembling, or repairing (involving partial disassembly and reassembly) the spacecraft, as required. If we exclude the possiblity of "irreversible" operations (that is, those caused by a force unknown to the robot, such as the expansion of a spring or an unexpected change in position of one or more parts due to gravity), then an assembly sequence may be thought of as the reverse of a disassembly sequence. This is the approach we have chosen; its advantages may be seen from the following rationale.

If the assembly task is approached directly, then the search's start state must be an arbitrary selection from an infinity of possible start states (all configurations of parts in which the spacecraft is disassembled), and the search must result in only one final state, that of completed assembly. The steps of the search are not guided by any obvious constraints; furthermore, the final state

may not be reachable from all possible start states, so that the completion of the search is not guaranteed. The situation is different when the disassembly task is considered. Here the start state is the assembled configuration, and each step of the search (individual part manipulations) is highly constrained. Although there are now an infinite number of final states (configurations of total disassembly) they may be linked by this criterion: determine an enclosing boundary (such as a cube) for the entire assembly; when all parts have been removed to positions outside this boundary, then an allowable final state has been reached. With the use of exhaustive depth-first search in the tree (modeled naturally by the Prolog backtracking construct) a solution is guaranteed to be found if it exists.

The method of the strategy planner, then, is to determine a partial or complete disassembly sequence, and reverse it if required. To make the search both possible and efficient, the algorithms mentioned above have been written. They are underlined in the following description of the execution of the search procedure, and are later described in detail.

1. Choose a direction for approach of the robot to the spacecraft based on principal axes, determine the rotation needed so that the chosen direction points to the conceptual front, and store the rotation information for later use in determining a reversed assembly list, if required.

2. Find the envelopes for each part remaining in the assembly based on the current direction of approach.

3. Determine the visibility of all parts from this direction, and try to remove those which are visible, frontmost first, using information stored as previous experience.

4. Find a gripping position on the first part whose removal is to be attempted using the appropriate tool, and determine whether a trajectory for the robot and tool from outside the enclosing boundary of the spacecraft to that position can be accomplished. Using the interference algorithm, treat the sections of tool and robot that will enter the boundary using the determined trajectory as a new "part", and verify that the tool and robot can approach and grip the part.

5. To determine whether or not the chosen part can be removed along the chosen trajectory, first use the extrusion algorithm and then the interference algorithm to see which, if any, of the other parts in the assembly will prevent this part's removal.

6. If the part can be removed successfully, perform the movement of the part to a position outside the boundary of the assembly and perform a conceptual removal of the part by retracting all references to it in the database. Continue by attempting to remove the next part.

7. If the part cannot be removed, store the reason for failure as previous experience and decide whether to attempt to remove the same part along a different trajectory, or to attempt to remove another part first.

8. Continue in this fashion until total disassembly is achieved, at which time reverse the procedure for disassembly to produce a procedure for assembly, if required.

It may be noted that the above search procedure is really a recursive search, since after each part removal, a new, smaller spacecraft remains to be disassembled. The end of the search occurs when the list of parts remaining to be removed is empty. The search is not purely recursive, however, because one of the heuristics involves recalling experience gained previous to the current search step.

Heuristics

Four heuristic algorithms are employed to guide and limit the search:

Principal Axes

Envelopes

Visibility

Previous Experience

These algorithms help to make each choice in the search procedure a good or at least reasonable one, since at many points an infinity of possibilities is present.

The principal axes of a part are those vectors that form significant direction lines in the part and along which it is likely to be possible to remove the part from the entire spacecraft. For example, a bolt represented as a cylinder has as its principal axis the vector down the center of the cylinder, and is most likely to be removed along the trajectory that is this vector. This heuristic is also used to choose directions along which the robot should approach the spacecraft; a direction that is one of the principal axes of many parts is better than one of only a few.

Principal axes are determined for each part by the following procedure:

1. get the vector of extrusion for each solid of linear extrusion, get the axis of revolution for each solid of revolution, get the lines of a polygon, and get the bisecting line(s) of an ellipse, parabola, or hyperbola

2. order the resulting list of axes by length (longest first and most likely) and number (most axes in the same direction; a plate with many bolt holes will have many axes in the direction of the cylindrical holes and will most likely be removed along that direction)

The procedure results in at least one axis for each part, but it does not mean that removal is possible along that axis. If necessary, axes may also be determined from the envelope of a part.

Part envelopes are simply rectangular boxes that completely enclose each part and are always oriented along the current Cartesian coordinates. The collection of envelopes thus becomes a rough "blocks world" approximation of the spacecraft. Since computations with envelopes are much simpler and faster than computations with the original parts, the envelopes are very useful in determining first approximations to part choices and removal strategies, and in limiting the number of interactions that must be investigated between actual parts.

Envelopes are stored for each part as the greatest and least extent in each of the three Cartesian coordinates. They are calculated by the following algorithm:

1. If the part is the union of two subparts, get the envelope of each subpart and take, for each coordinate, the greatest of the greater and the least of the lesser value.

2. If the part is the intersection of two subparts, get the envelope of each subpart and take, for each coordinate, the least of the greater and the greatest of the lesser value.

3. If the part is the difference of a subpart and a primitive, get the envelope of each and determine the correct overall envelope based on the relative positioning of the primitive within the subpart (usually the envelope is the same as the subpart's).

4. For a solid of linear extrusion, first translate the solid so that a convenient point in the planar curve is located on the origin of a Cartesian coordinate system. Now rotate the solid so that the vector of extrusion is collinear with the X-axis. Determine the envelope by taking the dimensions of a rectangle circumscribed around the planar curve in the YZ plane and the length of the vector of extrusion. Now rotate the envelope back to the original orientation of the part and determine the new maximal and minimal values for each of the three coordinates. Finally, reverse the first translation and the envelope has been calculated.

5. For a solid of revolution, first translate the solid so that the axis of revolution intersects the origin of a Cartesian coordinate system. Second, rotate the solid so that the axis of revolution is collinear with the X-axis. Third, translate the solid again so that a convenient point of the planar curve is on the Z-axis. Determine maximal and minimal points of intersection on the Z and X axes (the Y axis is the same as the Z) and as above, relocate the envelope to the original part position.

Once envelopes have been calculated for all parts, the visibility heuristic may be determined. If a human were given the task of attempting to remove parts from a given face of an assembly, he would of course use vision to decide which parts were accessible for touching, and which of these were closest to him and therefore good candidates for removal. The visibility heuristic mimics these attributes of vision by determining which part envelopes are not obstructed by other part envelopes from a given direction. The algorithm works as follows:

1. Order the envelopes based on greatest extent in the chosen direction (so that the frontmost envelope is first).

2. Starting with the frontmost envelope, determine which envelope faces are completely obstructed, and hence, invisible. If an envelope face is only partially obstructed, divide the remainder into rectangles and continue the process.

3. The result is a list of part envelopes with visible faces, ordered from the front.

The last heuristic used in the search procedure is that of previous experience. If a chosen part cannot be removed along a chosen trajectory, the precise reason for the failure (as determined by the interference algorithm) is marked in the database. If another search step should be tried involving this part, the information stored will be used again to limit the search.

For example, consider a simple peg composed of two cylinders in a block with a hole. If an attempt is made to move the peg sideways, the lower cylinder will interfere with the block. A notation is made of this result, such as "subpart_4 block_1 (0.15E+01,0.0,0.0)" indicating that interference occurred between two primitives when the first was moved along a particular trajectory. This information can be used in a number of ways: this particular combination should not be tried again unless one or both parts have first been moved in other ways; shorter trajectories in the same direction are probably not as good candidates as are trajectories in other directions; any manipulation of higher-level parts of which these primitives are subparts should investigate the interaction of these subparts first, as they are known to have caused interference before.

Algorithms of Feasibility

These are the algorithms that decide whether or not a given search step can be successfully executed. There are numerous problems that may be encountered if an arbitrary part is proposed for removal along an arbitrary trajectory. The geometry of the part itself may prevent removal in this fashion, other parts may need to be removed first (such as is the case with a bolted plate), or one or more other parts may be in the path of the proposed trajectory.

To treat all of these cases we use a combination of two algorithms - extrusion and interference. When a proposed part-trajectory pair has been chosen, the extrusion algorithm

calculates the greatest cross-sectional area of the part along the axis defined by the trajectory vector. This area now becomes a planar curve and the trajectory vector an extrusion vector in the creation of a new conceptual solid of linear extrusion. This new "part" contains all of the three-dimensional points that will be occupied by the actual part as it passes along the removal trajectory -- the entire volume swept during removal.

After the new solid defining the extruded original part has been determined, the interference algorithm operates. This algorithm decides whether or not the new solid "interferes" with any actual part in the geometric database; interference occurs when two solids occupy the same space. If any interference is encountered, then the chosen part cannot be removed along the proposed trajectory, as it would run into another part during the process.

As a first check, the interference algorithm is run on the enclosing envelopes of the parts to determine which, if any, parts need to be more closely investigated. Following this, the algorithm takes the description of each actual part in turn and compares it to the description of the new extruded "part". If an interference is found, the proposed removal is deemed unsuccessful, and the reason for failure is stored in the database.

The interference algorithm operates at its basic level with solids of linear extrusion formed from convex curves. Because these space-filling objects are also convex, it is possible to determine whether or not they have any points in common (they cannot

"intertwine" or be "hidden" within each other). The tree structure of Boolean operators combines these primitives to form complex parts. The interference algorithm decomposes them as follows:

1. If the parts are both primitives, compare them for common points.

2. If one or both parts are defined by the union operator, make comparisons between each pair of subparts. A case of interference in either defining subpart will also indicate an interference with the union.

Algorithms of Manipulation

There are three of these algorithms; they are needed to effect changes to the geometric database as the search proceeds and as parts are removed:

Rotation

Movement

Removal

For simplicity in calculations, removal trajectories are always assumed to be on the line from the current front face of the spacecraft towards the robot, and the robot arm is assumed to be fixed in this location. (This need not actually be the case; it is only assumed for convenience). This means that the strategy planner must be able to rotate its view of the database and so be able to approach

it from any angle in space. The rotation algorithm accomplishs this task. Starting from the top of the tree structure in the database (the "partlist"), it rotates each part, subpart, and planar defining curve in turn.

When a successful part-trajectory removal pair has been determined, the movement algorithm calculates the actual path of motion for the part from its assembled location to its new location outside the boundary of the spacecraft. The conceptual removal of the part from the database is performed by the removal algorithm which, beginning from the top of the part definition, retracts all references to it and its constituent defining entities, together with any notations that may have been made about it as previous experience.

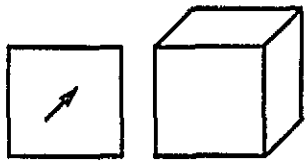
Conclusions and Future Work

The strategy planner described in this report has been designed and partially implemented in Prolog with the goal of automatically producing plans of robot commands for the assembly, disassembly, or repair of NASA spacecraft hardware. Successful results have been obtained for assembly and disassembly sequences for several test cases, including the blocks model with connecting bolts described in Appendix A. The chosen knowledge representation has shown itself to be well-suited to the task, and the fundamental algorithms have proven to be useful and efficient.

Future work on the strategy planner will involve extending existing Prolog code to cover the universe of options already designed, including the following specific objectives:

- operate on solids of revolution
- allow the use of conic arcs in planar curve definitions
- incorporate robot and tool information to include the calculation of tool placement and robot motion trajectories to the basic part trajectories
- allow curved and multi-path part trajectories
- extend the interference algorithm to operate correctly with the Boolean intersection and difference operators
- provide an algorithm to automatically decompose composite, concave planar curves into simple, convex curves
- incorporate lists of part attributes as special knowledge to further aid the search procedure
- develop a more sophisticated visibility heuristic that operates on actual parts rather than on envelopes

Figures



```
lexsolid(curve_1,1.0,1.0,1.0)
curve_1(polygon([line_1,line_2,line_3,line_4]))
line_1(start(1.0,0.0,0.0),end(0.0,0.0,0.0))
```

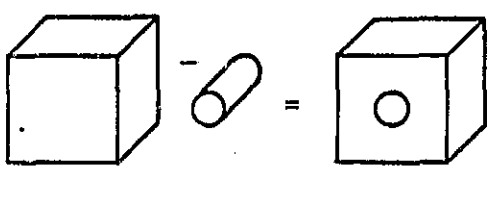
Figure 1

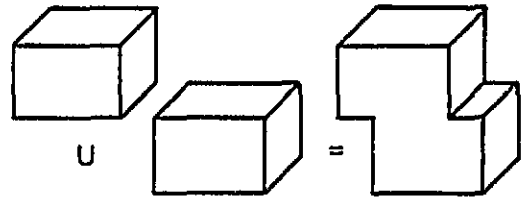
This figure displays a picture of a solid of linear extrusion (lexsolid) together with the actual Prolog code which defines it in the database. (Only line_1 is shown; the other lines are defined similarly).

```
revsolid(line_5,curve_2,0.0,0.0)
```

Figure 2

A cone defined as a solid of revolution (revsolid). Equal starting and ending angles indicate that one complete revolution is desired.






```
block_1(difference(subblock_1,hole_1))
```

```
block_2(union(subblock_2,subblock_3))
```

Figure 3

Two parts defined as primitives connected by Boolean operators. Note that each of these parts could in turn become subparts within other definitions.

```
partlist([lbolt_1,base,sbolt_2,...]).

base(difference(subpart_1,bolthole_1),
        envelope(1.0,3.1,4.0,8.0,3.2,6.0),
        attributes(...)).

bolthole_1(revsolid(line_1,curve_2,1.0,3.0),
        envelope(-11.0,2.0,3.0,-1.0,5.0,6.0)).

subpart_1(lexsolid(curve_1,1.0,1.0,3.0),
        envelope(1.0,2.0,1.0,3.0,4.0,3.0)).

curve_2(circarc(center(0.0,0.0,0.0),radius(5.0))).

curve_1(polygon([line_2,line_3,line_4])).

line_1(start(1.0,0.0,0.0),end(5.0,1.0,0.0)).
```

Figure 4

A sample implementation of the context-free grammar representation for space-filling objects. Note the use of Boolean operators in a tree structure, the use of unique names for each entity, and the inclusion of envelopes for parts and subparts and attributes for parts.

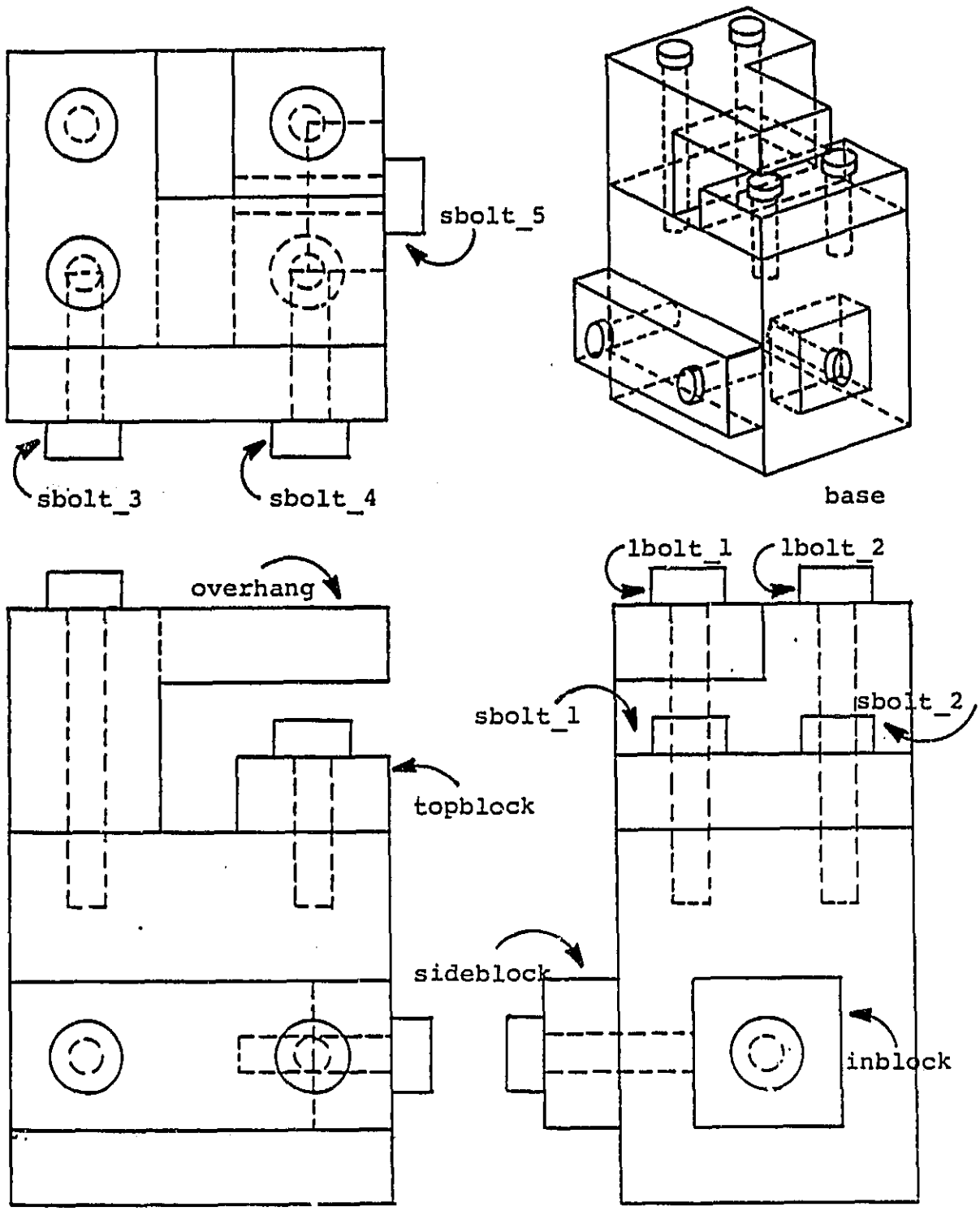


Figure 5

Mechanical Drawing of Blocks Model

Appendix A: Blocks Model Example

As an example of the actual use of the strategy planner, we present some of the internal data generated during operation on a blocks model designed for the purpose of testing the strategy planner. Appendix C presents the actual output from the strategy planner for the entire model.

Figure 5 shows four different views of the blocks model, and labels each piece with the name used by the strategy planner to distinguish parts. The primitives used are all solids of linear extrusion; bolts are composed of the union of two cylinders and the blocks are created from the difference of rectangular solids and cylinders representing the bolt holes.

Following the descriptions of the search procedure together with associated algorithms given above, the strategy planner works in this manner:

1. A determination of principal axes of each part is made; an analysis of the length and relative numbers of these axes indicates that the top of the model will be a promising face to first approach with the robot.

2. The rotation needed to bring the top of the model to the front is calculated and stored.

3. The visibility algorithm is run (using envelopes), resulting in a list of visible parts from the front ordered by proximity to the robot: (lbolt_1, lbolt_2, overhang, sbolt_2, topblock, base, sideblock, sbolt_3, sbolt_4, sbolt_5). Note that "inblock" and "sbolt_1" do not appear.

4. The first part in this list is chosen as a good candidate for removal. Thus "lbolt_1" is extruded along its principal axis (towards the front) and the interference algorithm is run with the new extruded part and neighboring parts.

5. Having determined that "lbolt_1" can in fact be removed, all references to it in the database are retracted by the removal algorithm, and the removal trajectory is calculated and stored.

6. The strategy planner continues in this fashion, removing successfully in turn "lbolt_2", "overhang", and "sbolt_2". Note that after each removal the visibility algorithm is run again, so that when "overhang" is gone, "sbolt_1" will become visible and hence a candidate for removal.

7. After the planner has exhausted the possibilities from the first direction of approach (it will have unsuccessfully attempted to remove "sideblock" and "sbolt_3, sbolt_4, sbolt_5") it goes to the next best principal axis, rotates, and begins again.

8. When all parts have been removed, the information stored during execution is concatenated into a disassembly list. If required, this list is reversed step by step into an assembly list.

Appendix B: IGES and Context-Free Grammar Part Specifications

This example displays first the IGES and then the context-free grammar representation for the part labeled "sideblock" in the blocks model of Figure 5.

The IGES representation is divided into two parts: the directory section, with entries labeled "D"; and the parameter section, with entries labeled "P". Directory entries serve as pointers into the parameter section, where numerical values are stored. Parameter entries are coded by number: 408 is an instance of an entity described under 308; 124 is a transformation matrix which may rotate and translate an entity in space; 180 is the entry storing Boolean operator information; the other entries store geometric entities.

124	1	1	1	0	0	0	D	1
124	0	0	2	0	0	0	D	2
164	3	1	1	0	0	1	D	3
164	0	0	1	0	0	0	D	4
124	4	1	1	0	0	0	D	5
124	0	0	2	0	0	0	D	6
100	6	1	1	0	0	5	D	7
100	0	0	1	0	0	0	D	8
308	7	1	1	0	0	0	D	9
308	0	0	1	0	0	0	D	10
124	69	1	1	0	0	0	D	103
124	0	0	2	0	0	0	D	104
164	71	1	1	0	0	103	D	105
164	0	0	1	0	0	0	D	106
102	72	1	1	0	0	0	D	107
102	0	0	1	0	0	0	D	108
110	73	1	1	0	0	0	D	109
110	0	0	1	0	0	0	D	110
110	74	1	1	0	0	0	D	111
110	0	0	1	0	0	0	D	112
110	75	1	1	0	0	0	D	113
110	0	0	1	0	0	0	D	114
110	76	1	1	0	0	0	D	115
110	0	0	1	0	0	0	D	116

```
124       77        1        1        0        0        0        D     117
124        0        0        2        0        0        0        D     118
408       79        1        1        0        0      117        D     119
408        0        0        1        0        0        0        D     120
124       80        1        1        0        0        0        D     121
124        0        0        2        0        0        0        D     122
408       82        1        1        0        0      121        D     123
408        0        0        1        0        0        0        D     124
180       83        1        1        0        0        0        D     125
180        0        0        1        0        0        0        D     126
308       84        1        1        0        0        0        D     127
308        0        0        1        0        0        0        D     128
124      160        1        1        0        0        0        D     255
124        0        0        2        0        0        0        D     256
408      162        1        1        0        0      255        D     257
408        0        0        1        0        0        0        D     258
```

```
124,1.000,0.0,0.0,0.0,0.0,1.000,0.0,0.0,0.0,0.0,1.000,          1P   1
0.0,0,0;                                                        1P   2
164,7,0.0,-1.000,0.0,0,0;                                       3P   3
124,1.000,0.0,0.0,0.0,0.0,0.0,1.000,0.0,0.0,                    5P   4
-1.000,0.0,0.0,0,0;                                             5P   5
100,0.0,0.0,0.0,0.2500,0.0,0.2500,0.0,0,0;                      7P   6
308,0,BHOLE1,1,3,0,0;                                           9P   7
124,1.000,0.0,0.0,0.0,0.0,1.000,0.0,0.0,0.0,0.0,1.000,        103P  69
0.0,0,0;                                                      103P  70
164,107,0.0,0.0,2.000,0,0;                                    105P  71
102,4,109,111,113,115,0,0;                                    107P  72
110,0.0,0.0,0.0,5.000,0.0,0.0,0,0;                            109P  73
110,5.000,0.0,0.0,5.000,1.000,0.0,0,0;                        111P  74
110,5.000,1.000,0.0,0.0,1.000,0.0,0,0;                        113P  75
110,0.0,1.000,0.0,0.0,0.0,0.0,0,0;                            115P  76
124,1.000,0.0,0.0,0.0,0.0,1.000,0.0,0.0,0.0,0.0,1.000,        116P  77
0.0,0,0;                                                      116P  78
408,9,1.000,1.000,1.000,1.000,0,0;                            119P  79
124,1.000,0.0,0.0,0.0,0.0,1.000,0.0,0.0,0.0,0.0,1.000,        120P  80
0.0,0,0;                                                      120P  81
408,9,4.000,1.000,1.000,1.000,0,0;                            123P  82
180,2,3,2,5,2,3,105,119,123,0,0;                              125P  83
308,1,SBLOCK,4,105,119,123,125,0,0;                           127P  84
124,1.000,0.0,0.0,0.0,0.0,0.0,-1.000,0.0,0.0,                 254P 160
1.000,0.0,0.0,0,0;                                            254P 161
408,127,0.0,3.000,4.000,1.000,0,0;                            257P 162
```

The context-free grammar representation explicitly names each entity in the part description. Note that the Boolean data is intrinsically incorporated.

```
sblock_1(difference(subpart_10,bholel_9)).

subpart_10(difference(subpart_9,bholel_8)).

subpart_9(lexsolid(curve_10,0.00E+00,-(0.20E+01),
       0.00E+00)).

curve_10(polygon([line_9,line_10,line_11,line_12])).

line_9(start(0.00E+00,0.30E+01,0.40E+01),
       end(0.50E+01,0.30E+01,0.40E+01)).

line_10(start(0.50E+01,0.30E+01,0.40E+01),
       end(0.50E+01,0.30E+01,0.50E+01)).

line_11(start(0.50E+01,0.30E+01,0.50E+01),
       end(0.00E+00,0.30E+01,0.50E+01)).

line_12(start(0.00E+00,0.30E+01,0.50E+01),
       end(0.00E+00,0.30E+01,0.40E+01)).

bholel_8(lexsolid(curve_11,0.00E+00,0.00E+00,
       -(0.10E+01))).

curve_11(circarc(center(0.10E+01,0.20E+01,0.50E+01),
       radius(0.25E+00))).

bholel_9(lexsolid(curve_12,0.00E+00,0.00E+00,
       -(0.10E+01))).

curve_12(circarc(center(0.40E+01,0.20E+01,0.50E+01),
       radius(0.25E+00))).
```

Appendix C: Strategy Planner Output

Listed is the output of the strategy planner when run on the blocks model of Figure 5 and Appendix A.

The requested operation was assembly; for each individual part, a vector and part description are given. The description locates the part at its initial position. When moved along the preceding vector, the part will be correctly assembled into the device.

```
(-(0.525E+01),0.00E+00,0.00E+00)

base_1(difference(subpart_8,ninblock_1)).
subpart_8(difference(subpart_7,bholel_7)).
subpart_7(difference(subpart_6,bholel_6)).
subpart_6(difference(subpart_5,bholel_5)).
subpart_5(difference(subpart_4,bholel_4)).
subpart_4(difference(subpart_3,bholel_3)).
subpart_3(difference(subpart_2,bholel_2)).
subpart_2(difference(subpart_1,bholel_1)).
subpart_1(lexsolid(curve_1,0.00E+00,0.00E+00,
        0.40E+01)).
curve_1(polygon([line_1,line_2,line_3,line_4])).
line_1(start(0.525E+01,0.00E+00,0.00E+00),
        end(0.1025E+02,0.00E+00,0.00E+00)).
line_2(start(0.1025E+02,0.00E+00,0.00E+00),
        end(0.1025E+02,0.50E+01,0.00E+00)).
line_3(start(0.1025E+02,0.50E+01,0.00E+00),
        end(0.525E+01,0.50E+01,0.00E+00)).
line_4(start(0.525E+01,0.50E+01,0.00E+00),
        end(0.525E+01,0.00E+00,0.00E+00)).
bholel_1(lexsolid(curve_2,0.00E+00,-(0.10E+01),
        0.00E+00)).
curve_2(circarc(center(0.625E+01,0.50E+01,0.10E+01),
        radius(0.25E+00))).
bholel_2(lexsolid(curve_3,0.00E+00,-(0.10E+01),
        0.00E+00)).
curve_3(circarc(center(0.625E+01,0.50E+01,0.30E+01),
        radius(0.25E+00))).
bholel_3(lexsolid(curve_4,0.00E+00,-(0.10E+01),
        0.00E+00)).
curve_4(circarc(center(0.925E+01,0.50E+01,0.10E+01),
        radius(0.25E+00))).
bholel_4(lexsolid(curve_5,0.00E+00,-(0.10E+01),
```

```
        0.00E+00)).
curve_5(circarc(center(0.925E+01,0.50E+01,0.30E+01),
        radius(0.25E+00))).
bholel_5(lexsolid(curve_6,0.00E+00,0.00E+00,
        -(0.10E+01))).
curve_6(circarc(center(0.625E+01,0.20E+01,0.40E+01),
        radius(0.25E+00))).
bholel_6(lexsolid(curve_7,0.00E+00,0.00E+00,
        -(0.10E+01))).
curve_7(circarc(center(0.925E+01,0.20E+01,0.40E+01),
        radius(0.25E+00))).
bholel_7(lexsolid(curve_8,-(0.10E+01),0.00E+00,
        0.00E+00)).
curve_8(circarc(center(0.925E+01,0.20E+01,0.20E+01),
        radius(0.25E+00))).
ninblock_1(lexsolid(curve_9,0.00E+00,-(0.20E+01),
        0.00E+00)).
curve_9(polygon([line_5,line_6,line_7,line_8])).
line_5(start(0.925E+01,0.30E+01,0.30E+01),
        end(0.925E+01,0.30E+01,0.10E+01)).
line_6(start(0.925E+01,0.30E+01,0.10E+01),
        end(0.1025E+02,0.30E+01,0.10E+01)).
line_7(start(0.1025E+02,0.30E+01,0.10E+01),
        end(0.1025E+02,0.30E+01,0.30E+01)).
line_8(start(0.1025E+02,0.30E+01,0.30E+01),
        end(0.925E+01,0.30E+01,0.30E+01)).


(-(0.125E+01),0.00E+00,0.00E+00)

inblock_1(difference(subpart_11,bholel_10)).
subpart_11(lexsolid(curve_13,0.00E+00,-(0.20E+01),
        0.00E+00)).
curve_13(polygon([line_13,line_14,line_15,line_16])).
line_13(start(0.525E+01,0.30E+01,0.30E+01),
        end(0.525E+01,0.30E+01,0.10E+01)).
line_14(start(0.525E+01,0.30E+01,0.10E+01),
        end(0.625E+01,0.30E+01,0.10E+01)).
line_15(start(0.625E+01,0.30E+01,0.10E+01),
        end(0.625E+01,0.30E+01,0.30E+01)).
line_16(start(0.625E+01,0.30E+01,0.30E+01),
        end(0.525E+01,0.30E+01,0.30E+01)).
bholel_10(lexsolid(curve_14,-(0.10E+01),0.00E+00,
        0.00E+00)).
curve_14(circarc(center(0.625E+01,0.20E+01,0.20E+01),
        radius(0.25E+00))).


(-(0.225E+01),0.00E+00,0.00E+00)

sbolt_5(union(subpart_29,subpart_30)).
subpart_29(lexsolid(curve_34,0.25E+00,0.00E+00,
```

```
        0.00E+00)).
curve_34(circarc(center(0.725E+01,0.20E+01,0.20E+01),
        radius(0.375E+00))).
subpart_30(lexsolid(curve_35,-(0.20E+01),0.00E+00,
        0.00E+00)).
curve_35(circarc(center(0.725E+01,0.20E+01,0.20E+01),
        radius(0.25E+00))).


(0.00E+00,0.00E+00,-(0.125E+01))

sblock_1(difference(subpart_10,bholel_9)).
subpart_10(difference(subpart_9,bholel_8)).
subpart_9(lexsolid(curve_10,0.00E+00,-(0.20E+01),
        0.00E+00)).
curve_10(polygon([line_9,line_10,line_11,line_12])).
line_9(start(0.00E+00,0.30E+01,0.525E+01),
        end(0.50E+01,0.30E+01,0.525E+01)).
line_10(start(0.50E+01,0.30E+01,0.525E+01),
        end(0.50E+01,0.30E+01,0.625E+01)).
line_11(start(0.50E+01,0.30E+01,0.625E+01),
        end(0.00E+00,0.30E+01,0.625E+01)).
line_12(start(0.00E+00,0.30E+01,0.625E+01),
        end(0.00E+00,0.30E+01,0.525E+01)).
bholel_8(lexsolid(curve_11,0.00E+00,0.00E+00,
        -(0.10E+01))).
curve_11(circarc(center(0.10E+01,0.20E+01,0.625E+01),
        radius(0.25E+00))).
bholel_9(lexsolid(curve_12,0.00E+00,0.00E+00,
        -(0.10E+01))).
curve_12(circarc(center(0.40E+01,0.20E+01,0.625E+01),
        radius(0.25E+00))).


(0.00E+00,0.00E+00,-(0.225E+01))

sbolt_4(union(subpart_27,subpart_28)).
subpart_27(lexsolid(curve_32,0.00E+00,0.00E+00,
        0.25E+00)).
curve_32(circarc(center(0.40E+01,0.20E+01,0.725E+01),
        radius(0.375E+00))).
subpart_28(lexsolid(curve_33,0.00E+00,0.00E+00,
        -(0.20E+01))).
curve_33(circarc(center(0.40E+01,0.20E+01,0.725E+01),
        radius(0.25E+00))).


(0.00E+00,0.00E+00,-(0.225E+01))
sbolt_3(union(subpart_25,subpart_26)).
subpart_25(lexsolid(curve_30,0.00E+00,0.00E+00,
        0.25E+00)).
curve_30(circarc(center(0.10E+01,0.20E+01,0.725E+01),
```

```
        radius(0.375E+00))).
subpart_26(lexsolid(curve_31,0.00E+00,0.00E+00,
        -(0.20E+01))).
curve_31(circarc(center(0.10E+01,0.20E+01,0.725E+01),
        radius(0.25E+00))).


(0.00E+00,-(0.325E+01),0.00E+00)

topblock_1(difference(subpart_13,bholel_12)).
subpart_13(difference(subpart_12,bholel_11)).
subpart_12(lexsolid(curve_15,0.20E+01,0.00E+00,
        0.00E+00)).
curve_15(polygon([line_17,line_18,line_19,line_20])).
line_17(start(0.30E+01,0.825E+01,0.40E+01),
        end(0.30E+01,0.825E+01,0.00E+00)).
line_18(start(0.30E+01,0.825E+01,0.00E+00),
        end(0.30E+01,0.925E+01,0.00E+00)).
line_19(start(0.30E+01,0.925E+01,0.00E+00),
        end(0.30E+01,0.925E+01,0.40E+01)).
line_20(start(0.30E+01,0.925E+01,0.40E+01),
        end(0.30E+01,0.825E+01,0.40E+01)).
bholel_11(lexsolid(curve_16,0.00E+00,-(0.10E+01),
        0.00E+00)).
curve_16(circarc(center(0.40E+01,0.925E+01,0.30E+01),
        radius(0.25E+00))).
bholel_12(lexsolid(curve_17,0.00E+00,-(0.10E+01),
        0.00E+00)).
curve_17(circarc(center(0.40E+01,0.925E+01,0.10E+01),
        radius(0.25E+00))).


(0.00E+00,-(0.425E+01),0.00E+00)

sbolt_1(union(subpart_21,subpart_22)).
subpart_21(lexsolid(curve_26,0.00E+00,0.25E+00,
        0.00E+00)).
curve_26(circarc(center(0.40E+01,0.1025E+02,0.10E+01),
        radius(0.375E+00))).
subpart_22(lexsolid(curve_27,0.00E+00,-(0.20E+01),
        0.00E+00)).
curve_27(circarc(center(0.40E+01,0.1025E+02,0.10E+01),
        radius(0.25E+00))).


(0.00E+00,-(0.425E+01),0.00E+00)

sbolt_2(union(subpart_23,subpart_24)).
subpart_23(lexsolid(curve_28,0.00E+00,0.25E+00,
        0.00E+00)).
curve_28(circarc(center(0.40E+01,0.1025E+02,0.30E+01),
        radius(0.375E+00))).
```

```
subpart_24(lexsolid(curve_29,0.00E+00,-(0.20E+01),
        0.00E+00)).
curve_29(circarc(center(0.40E+01,0.1025E+02,0.30E+01),
        radius(0.25E+00))).


(0.00E+00,-(0.325E+01),0.00E+00)
overhang_1(difference(subpart_16,ohhole1_2)).
subpart_16(difference(subpart_15,ohhole1_1)).
subpart_15(union(subpart_14,ohl_1)).
subpart_14(lexsolid(curve_18,0.00E+00,-(0.30E+01),
        0.00E+00)).
curve_18(polygon([line_21,line_22,line_23,line_24])).
line_21(start(0.00E+00,0.1125E+02,0.40E+01),
        end(0.00E+00,0.1125E+02,0.00E+00)).
line_22(start(0.00E+00,0.1125E+02,0.00E+00),
        end(0.20E+01,0.1125E+02,0.00E+00)).
line_23(start(0.20E+01,0.1125E+02,0.00E+00),
        end(0.20E+01,0.1125E+02,0.40E+01)).
line_24(start(0.20E+01,0.1125E+02,0.40E+01),
        end(0.00E+00,0.1125E+02,0.40E+01)).
ohl_1(lexsolid(curve_19,0.00E+00,-(0.10E+01),
        0.00E+00)).
curve_19(polygon([line_25,line_26,line_27,line_28])).
line_25(start(0.20E+01,0.1125E+02,0.40E+01),
        end(0.20E+01,0.1125E+02,0.20E+01)).
line_26(start(0.20E+01,0.1125E+02,0.20E+01),
        end(0.50E+01,0.1125E+02,0.20E+01)).
line_27(start(0.50E+01,0.1125E+02,0.20E+01),
        end(0.50E+01,0.1125E+02,0.40E+01)).
line_28(start(0.50E+01,0.1125E+02,0.40E+01),
        end(0.20E+01,0.1125E+02,0.40E+01)).
ohhole1_1(lexsolid(curve_20,0.00E+00,0.30E+01,
        0.00E+00)).
curve_20(circarc(center(0.10E+01,0.825E+01,0.30E+01),
        radius(0.25E+00))).
ohhole1_2(lexsolid(curve_21,0.00E+00,0.30E+01,
        0.00E+0.00).
curve_21(circarc(center(0.10E+01,0.825E+01,0.10E+01),
        radius(0.25E+00))).


(0.00E+00,-(0.425E+01),0.00E+00)

lbolt_2(union(subpart_19,subpart_20)).
subpart_19(lexsolid(curve_24,0.00E+00,0.25E+00,
        0.00E+00)).
curve_24(circarc(center(0.10E+01,0.1225E+02,0.30E+01),
        radius(0.375E+00))).
subpart_20(lexsolid(curve_25,0.00E+00,-(0.40E+01),
        0.00E+00)).
curve_25(circarc(center(0.10E+01,0.1225E+02,0.30E+01),
```

```
        radius(0.25E+00))).


(0.00E+00,-(0.425E+01),0.00E+00)

lbolt_1(union(subpart_17,subpart_18)).
subpart_17(lexsolid(curve_22,0.00E+00,0.25E+00,
        0.00E+00)).
curve_22(circarc(center(0.10E+01,0.1225E+02,0.10E+01),
        radius(0.375E+00))).
subpart_18(lexsolid(curve_23,0.00E+00,-(0.40E+01),
        0.00E+00)).
curve_23(circarc(center(0.10E+01,0.1225E+02,0.10E+01),
        radius(0.25E+00))).
```

Appendix D: Sample Prolog Code

We give two examples of strategy planner algorithms implemented in Prolog.

The first is for part or device rotation in space. The input arguments are a direction and a notation for forward or backward rotation (referring to the order of application of the three angular rotations about the three Cartesian coordinates). Note that the rules successively decompose each part into its constituent entities and rotate each by removing the old description from the database and inserting the new, rotated version.

```
rotate_device([X,0.0,0.0],_) :-
        X>=0.0,
        partlist(Parts),
        rotate_part(Parts,[]).

rotate_device(Dir,forward) :-
        Dir=[X,Y,Z],
        get_rot_mat(X,Y,Z,Rot_mat),
        make_trans_mat(Rot_mat,[],Trans_mat),
        partlist(Parts),
        tell(rotation),
        rotate_part(Parts,Trans_mat),
        told.

rotate_device(Dir,backward) :-
        Dir=[X,Y,Z],
        get_rot_mat(X,Y,Z,Rot_mat),
        transpose_rot_mat(Rot_mat,New_mat),
        make_trans_mat(New_mat,[],Trans_mat),
        partlist(Parts),
        tell(rotation),
        rotate_part(Parts,Trans_mat),
        told.

rotate_part([],_).

rotate_part([Part|Partlist],[]) :-
        Part_des=..[Part,Des,Env],
```

```
        retract(Part_des),
        New_partdes=..[Part,Des],
        assert(New_partdes),
        rotate_part(Partlist,[]).

rotate_part(_,[]).

rotate_part([Part|Partlist],Trans_mat) :-
        Part_des=..[Part,Des,Env],
        retract(Part_des),
        rotate_entity(Des,New_des,Trans_mat),
        New_partdes=..[Part,New_des],
        assert(New_partdes),
        write(' '),write(New_partdes),
        write('.'),nl,nl,
        rotate_part(Partlist,Trans_mat).

rotate_part([Part|Partlist],Trans_mat) :-
        Part_des=..[Part,Des],
        retract(Part_des),
        rotate_entity(Des,New_des,Trans_mat),
        New_partdes=..[Part,New_des],
        assert(New_partdes),
        write(' '),write(New_partdes),
        write('.'),nl,nl,
        rotate_part(Partlist,Trans_mat).

rotate_entity(union(Ent_1,Ent_2),union(Nent_1,Nent_2),
                Trans_mat) :-
        rotate_entity(Ent_1,Nent_1,Trans_mat),
        rotate_entity(Ent_2,Nent_2,Trans_mat).

rotate_entity(intersection(Ent_1,Ent_2),
                intersection(Nent_1,Nent_2),
                Trans_mat) :-
        rotate_entity(Ent_1,Nent_1,Trans_mat),
        rotate_entity(Ent_2,Nent_2,Trans_mat).

rotate_entity(difference(Ent_1,Ent_2),
                difference(Nent_1,Nent_2),
                Trans_mat) :-
        rotate_entity(Ent_1,Nent_1,Trans_mat),
        rotate_entity(Ent_2,Nent_2,Trans_mat).

rotate_entity(lexsolid(Curve,X,Y,Z),lexsolid(Curve,NX,NY,NZ),
                Trans_mat) :-
        rotate_entity(Curve,_,Trans_mat),
        rotatescale_vector(Trans_mat,X,Y,Z,NX,NY,NZ).

rotate_entity(circarc(center(X,Y,Z),R),
                circarc(center(NewX,NewY,NewZ),R),
                Trans_mat) :-
```

```
        rotatescale_vector(Trans_mat,X,Y,Z,NewX,NewY,NewZ).

rotate_entity(polygon(Linelist),polygon(Linelist),Trans_mat) :-
        rotate_entity(Linelist,_,Trans_mat).

rotate_entity(start(X,Y,Z),start(NewX,NewY,NewZ),Trans_mat) :-
        rotatescale_vector(Trans_mat,X,Y,Z,NewX,NewY,NewZ).

rotate_entity(end(X,Y,Z),end(NewX,NewY,NewZ),Trans_mat) :-
        rotatescale_vector(Trans_mat,X,Y,Z,NewX,NewY,NewZ).

rotate_entity([],_,_).

rotate_entity([Line|Linelist],_,Trans_mat) :-
        rotate_entity(Line,_,Trans_mat),
        rotate_entity(Linelist,_,Trans_mat).

rotate_entity(Entity,Entity,Trans_mat) :-
        Ent_des=..[Entity,Des],
        retract(Ent_des),
        rotate_entity(Des,New_des,Trans_mat),
        New_entdes=..[Entity,New_des],
        assert(New_entdes),
        write(' '),write(New_entdes),write('.'),nl,nl.

rotate_entity(Line,Line,Trans_mat) :-
        Line_des=..[Line,Start,End],
        retract(Line_des),
        rotate_entity(Start,New_start,Trans_mat),
        rotate_entity(End,New_end,Trans_mat),
        New_linedes=..[Line,New_start,New_end],
        assert(New_linedes),
        write(' '),write(New_linedes),write('.'),nl,nl.
```

The visibility algorithm returns a list of all parts visible from the angle at which the entire device is being viewed.

It operates on the previously calculated enclosing spatial part envelopes.

```
visibility(List) :-
        partlist(Parts),
        order_parts(Parts,Result),
        visible_parts([],_,_,Result,List).

order_parts([],[]).
```

```
order_parts([Part|Partlist],List) :-
        Part_des=..[Part,Des,Env],
        call(Part_des),
        Env=..[envelope,[_,X,LY,MY,LZ,MZ]],
        order_parts(Partlist,Newlist),
        update_vislist(Newlist,Part,X,[LY,MY,LZ,MZ],List).

update_vislist([],Part,X,Rect,[Part,X,Rect]).

update_vislist([Part_vis,X_vis,Rect_vis|List],Part,X,Rect,
                [Part,X,Rect,Part_vis,X_vis,Rect_vis|List]) :-
        X>=X_vis.

update_vislist([Part_vis,X_vis,Rect_vis|List],Part,X,Rect,
                [Part_vis,X_vis,Rect_vis|Newlist]) :-
        update_vislist(List,Part,X,Rect,Newlist).

visible_parts([],_,_,[],[]).

visible_parts([_|_],_,_,[],[]).

visible_parts([],_,_,[Part,X,Rect|List],[Part|Newlist]) :-
        visible_parts([Rect],X,[],List,Newlist).

visible_parts(Rectlist,PrecX,Preclist,[Part,X,Rect|List],
                Newlist) :-
        X=PrecX,
        is_obliterated(Rect,Preclist),
        visible_parts(Rectlist,X,Preclist,List,Newlist).

visible_parts(Rectlist,PrecX,Preclist,[Part,X,Rect|List],
                [Part|Newlist]) :-
        X=PrecX,
        visible_parts([Rect|Rectlist],X,Preclist,List,Newlist).

visible_parts(Rectlist,_,_,[Part,X,Rect|List],Newlist) :-
        is_obliterated(Rect,Rectlist),
        visible_parts(Rectlist,X,Rectlist,List,Newlist).

visible_parts(Rectlist,_,_,[Part,X,Rect|List],[Part|Newlist]) :-
        visible_parts([Rect|Rectlist],X,Rectlist,List,Newlist).

is_obliterated(Rect,Rectlist) :-
        obliterated(Rect,Rectlist,Ans),
        !,
        Ans=yes.

obliterated([[R1,R2,R3,R4]|Newlist],Rectlist,Ans) :-
        obliterated([R1,R2,R3,R4],Rectlist,Newans),
        continue(Newlist,Rectlist,Newans,Ans).

obliterated(Rect,[],no).
```

```
obliterated([],_,yes).

obliterated([RLY,RMY,RLZ,RMZ],[[OLY,OMY,OLZ,OMZ]|Rectlist],Ans) :-
        (RLY>=OMY; OLY>=RMY;
        RLZ>=OMZ; OLZ>=RMZ),
        obliterated([RLY,RMY,RLZ,RMZ],Rectlist,Ans).

obliterated([RLY,RMY,RLZ,RMZ],[[OLY,OMY,OLZ,OMZ]|_],yes) :-
        RLY>=OLY, RMY=<OMY,
        RLZ>=OLZ, RMZ=<OMZ.

obliterated(Rect,[Orect|Rectlist],Ans) :-
        divide_rect(Rect,Orect,Newrectlist),
        obliterated(Newrectlist,Rectlist,Ans).

divide_rect([RLY,RMY,RLZ,RMZ],[OLY,OMY,OLZ,OMZ],Rectlist) :-
        RLY=<OLY, RMY=<OMY,
        check_rect([RLY,OLY,RLZ,RMZ],R1),
        check_rect([OLY,RMY,RLZ,OLZ],R2),
        check_rect([OLY,RMY,OMZ,RMZ],R3),
        make_rect_list([R1,R2,R3],Rectlist).

divide_rect([RLY,RMY,RLZ,RMZ],[OLY,OMY,OLZ,OMZ],Rectlist) :-
        RLY=<OLY, OMY=<RMY,
        check_rect([RLY,OLY,RLZ,RMZ],R1),
        check_rect([OLY,OMY,RLZ,OLZ],R2),
        check_rect([OLY,OMY,OMZ,RMZ],R3),
        check_rect([OMY,RMY,RLZ,RMZ],R4),
        make_rect_list([R1,R2,R3,R4],Rectlist).

divide_rect([RLY,RMY,RLZ,RMZ],[OLY,OMY,OLZ,OMZ],Rectlist) :-
        OLY=<RLY, RMY=<OMY,
        check_rect([RLY,RMY,RLZ,OLZ],R1),
        check_rect([RLY,RMY,OMZ,RMZ],R2),
        make_rect_list([R1,R2],Rectlist).

divide_rect([RLY,RMY,RLZ,RMZ],[OLY,OMY,OLZ,OMZ],Rectlist) :-
        OLY=<RLY, OMY=<RMY,
        check_rect([RLY,OMY,RLZ,OLZ],R1),
        check_rect([OMY,RMY,RLZ,RMZ],R2),
        check_rect([RLY,OMY,OMZ,RMZ],R3),
        make_rect_list([R1,R2,R3],Rectlist).

check_rect([X,X,_,_],[]).

check_rect([_,_,X,X],[]).

check_rect([A,B,_,_],[]) :-
        B<A.

check_rect([_,_,A,B],[]) :-
        B<A.
```

```
check_rect(Rect,Rect).

make_rect_list([],[]).

make_rect_list([[]|Rest],List) :-
        make_rect_list(Rest,List).

make_rect_list([Rect|Rest],[Rect|List]) :-
        make_rect_list(Rest,List).

continue(A,B,no,no).

continue(A,B,yes,Ans) :-
        obliterated(A,B,Ans).
```

List of References

Bonney, M.C. et al. "Verifying Robot Programs for Collision Free Tasks." Developments in Robotics 1983. IFS Publications, Ltd.: Bedford, England. 1983.

Boyse, John W. "Interference Detection Among Solids and Surfaces." Communications of the ACM. Vol. 22, No. 1. January, 1979. 3-9.

Brady, et al., eds. Robot Motion. The MIT Press: Cambridge, Massachusetts. 1983. 473-498.

Clocksin, W.F. and Mellish, C.S. Programming in Prolog. Springer-Verlag: New York, New York. 1982.

Lozano-Perez, T. "Spatial Planning: A Configuration Space Approach." IEEE Transactions on Computers. Vol. C-32, No. 2. February, 1983. 108-120.

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

Nilsson, Nils J. Principles of Artificial Intelligence. Tioga Publishing Company: Palo Alto, California. 1980.

Premack, T. et al. "Design and Implementation of a Compliant Robot with Force-Feedback and Strategy Planning Software". NASA Technical Memorandum 86111. National Aeronautics and Space Administration -- Goddard Space Flight Center: Greenbelt, Maryland. 1984.

Requicha, A.A.G. "Representations for Rigid Solids: Theory, Methods, and Systems." Computing Surveys. Vol. 12, No. 4. December, 1980. 437-464.

Smith, Bradford M., et al. IGES Experimental Solids Proposal. Unpublished report from the National Bureau of Standards. 1984.

Smith, Bradford M., et al. Initial Graphics Exchange Specification (IGES) Version 2.0. U.S. National Bureau of Standards: Washington, D.C. 1983.

BIBLIOGRAPHIC DATA SHEET

1. Report No. CR 175319	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A Strategy Planner for NASA Robotics Applications		5. Report Date July 24, 1985 6. Performing Organization Code
7. Author(s) Steven S. Brodd		8. Performing Organization Report No. 8580504
9. Performing Organization Name and Address Science Applications Research 4400 Forbes Boulevard Lanham, MD 20706		10. Work Unit No. 11. Contract or Grant No. NAS5-28200
12. Sponsoring Agency Name and Address Code 731.4 Mechanical Engineering Branch National Aeronautics & Space Administration Greenbelt, MD 20771		13. Type of Report and Period Covered Contractor Report 14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Automatic strategy or task planning is an important element of robotics systems. A strategy planner under development at Goddard Space Flight Center automatically produces robot plans for assembly, disassembly, or repair of NASA spacecraft from computer-aided design descriptions of the individual parts of the spacecraft.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Robotics, Knowledge Engineering, Artificial Intelligence, Spatial Reasoning, Prolog, Automatic Task Planning	Unlimited Unclassified Category 63

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price*
Unclassified	Unclassified	48	

*For sale by the National Technical Information Service, Springfield, Virginia

GSFC 25-44 (10/77)