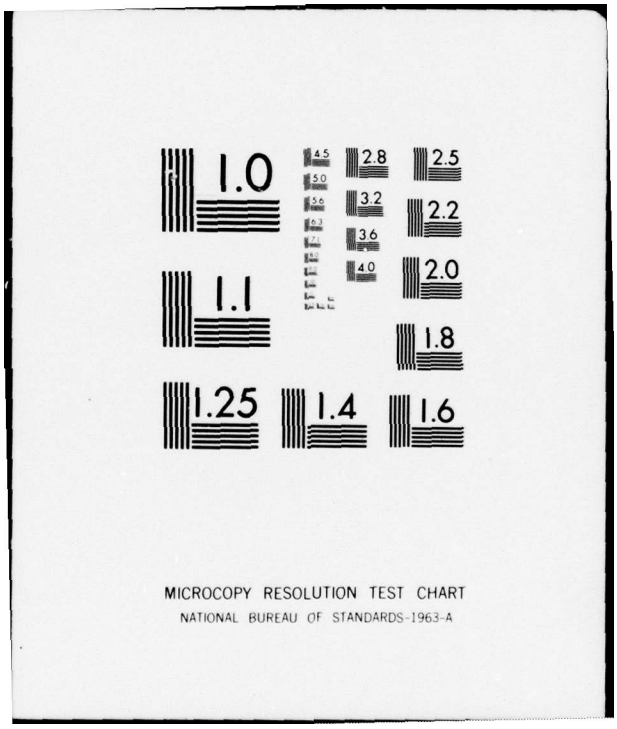
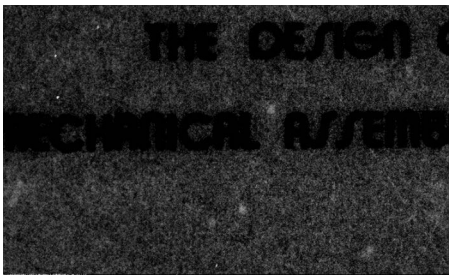
AD-A036 734 MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTE--ETC F/G 9/2
THE DESIGN OF A MECHANICAL ASSEMBLY SYSTEM.(U)
DEC 76 T LOZANO-PEREZ
UNCLASSIFIED A1-TR-397 N00014-75-C-0643
NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AI-TR-397	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: The Design of a Mechanical Assembly System	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Tomás Lozano-Pérez	8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0643
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, Massachusetts 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Blvd, Arlington, Virginia 22209	12. REPORT DATE: December 1976 13. NUMBER OF PAGES: 192
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Office of Naval Research, Information Systems, Arlington, Virginia 22217	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is unlimited.

188p.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Artificial Intelligence
Mechanical Assembly
Computer-controlled Manipulator
Spatial Modeling

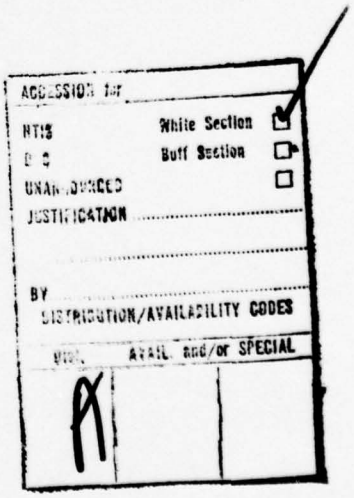
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This thesis describes a mechanical assembly system called LAMA. The goal of the work was to create a mechanical assembly system that transforms a high-level description of an automatic assembly operation into a program for execution by a computer-controlled manipulator. This system allows the initial description of the assembly to be in terms of the desired effects on the parts being assembled.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract N00014-75-C-0643.



THE DESIGN OF A MECHANICAL ASSEMBLY SYSTEM

by

Tomás Lozano-Pérez

Massachusetts Institute of Technology

December 1976

Revised version of a dissertation submitted to the Department of Electrical Engineering on December 20, 1976 in partial fulfillment of the requirements for the degree of Master of Science.

Abstract

This thesis describes a mechanical assembly system called LAMA (Language for Automatic Mechanical Assembly). The goal of the work was to create a mechanical assembly system that transforms a high-level description of an automatic assembly operation into a program for execution by a computer controlled manipulator. This system allows the initial description of the assembly to be in terms of the desired effects on the parts being assembled. Languages such as WAVE [Bolles & Paul] and MINI [Silver] fail to meet this goal by requiring the assembly operation to be described in terms of manipulator motions.

This research concentrates on the spatial complexity of mechanical assembly operations. The assembly problem is seen as the problem of achieving a certain set of geometrical constraints between basic objects while avoiding unwanted collisions. The thesis explores how these two facets, desired constraints and unwanted collisions, affect the primitive operations of the domain.

Three basic ideas underlie the design of LAMA:

(1) High level assembly operations can be represented by general program plans (called *skeletons*) which can be expanded as required by the details of individual assemblies.

(2) The desired effect of the basic manipulator motions can be described in terms of a few geometrical and spatial constraints.

(3) Choices for location and motion parameters should be made by identifying all the constraints on the solution, finding a legal range where the solution may lie and picking an element from the legal range.

Thesis Supervisor: Patrick H. Winston
Title: Associate Professor of Electrical Engineering

Acknowledgements

I would like to thank Prof. Patrick Winston for suggesting this topic of research and for supervising its progress. I would also like to thank him for many years of help and advice.

Prof. Berthold Horn read an earlier draft and was a constant source of encouragement.

I would like to thank Bob Woodham, Mark Lavin and Gene Freuder for their moral support.

Bob Woodham and Mark Lavin read drafts and provided much useful advice.

I would like to acknowledge a special debt of gratitude to Mark Lavin whose enthusiasm for the field sparked my own. Many of the ideas in this thesis are either due to him or originated in conversations with him.

David Silver contributed very generously of his time and expertise in an attempt to help me understand the use of manipulators.

Hirochika Inoue provided his insight into the problem during the early stages of this research.

I would also like to thank Mike Freiling, Mark Raibert and Shimon Ullman for their help.

My thanks to Karen Prendergast for the artwork in the figures and to Suzin Jabari for her help throughout.

Finally, I would like to thank Lorraine Gray for her support and perspective.

Contents

THE LAMA

by Ogden Nash

The one-l lama,
He's a priest.
The two-l llama,
He's a beast.
And I will bet
A silk pajama
There isn't any
Three-l lllama.

Figures

I. Introduction

This report describes a mechanical assembly system called **LAMA** (Language for Automatic Mechanical Assembly). The goal of the work is to create a system that transforms a high-level description of an automatic mechanical assembly operation into a program for execution by a computer controlled manipulator. This system allows the initial description of the assembly to be in terms of the desired effects on the parts being assembled. Languages such as **WAVE** [Bolles & Paul] and MINI [Silver] fail to meet this goal by requiring the assembly operation to be described in terms of manipulator motions.

The major goal of a very high-level language is to allow the user to ignore some of the detailed and relatively unimportant coding decisions. This goal tends to conflict with the requirement that the language be general. The result is that most very high-level languages are also very domain specific or problem-oriented. The domain specificity allows the language compiler (or interpreter) to embody a great deal of domain-dependent information and thus increase the number of decisions it can make for the user. The crucial aspect of the design of such a language is the analysis of the operations available in the domain. The decisions to be made must be identified and the basis for making them formalized. This report presents detailed analyses of many of the decisions that must be made when specifying an automatic mechanical assembly.

This research concentrates on the spatial complexity of mechanical assembly operations. The assembly problem is seen as the problem of achieving a certain set of geometrical constraints between basic objects while avoiding unwanted collisions. The report will explore how these two facets, desired constraints and unwanted collisions, affect the primitive operations of the

domain.

Three basic ideas underlie the design of LAMA:

(1) High level assembly operations can be represented by general program plans (called *skeletons*) which can be expanded as required by the details of individual assemblies.

(2) The desired effect of the basic manipulator motions can be described in terms of a few geometrical and spatial constraints.

(3) Choices for location and motion parameters should be made by identifying all the constraints on the solution, finding a range of values where the solution may lie and picking an element from that range.

The rest of this introductory chapter elaborates these ideas by first giving a sample assembly scenario, then defining how a total system might be structured into modules and how the modules can interact. Later, we will be specific about what modules have been implemented.

I.1 Assembling a Piston

Since we want to reduce the problems of programming assembly operations, the first step is to identify those problems. This section tries to highlight them by examining a particular assembly task in some detail.

Figure I.1 shows the piston subassembly from an ENYA 15-III model aircraft engine. The assembly is carried out using the Little Robot system at the MIT Artificial Intelligence Laboratory [Silver]. This manipulator is not a fully general position and orientation generator because it has only five degrees of freedom, not six. They are divided in the following manner: (1) an xy table, (2) a wrist which can displace and rotate along the z axis and (3) a vise which

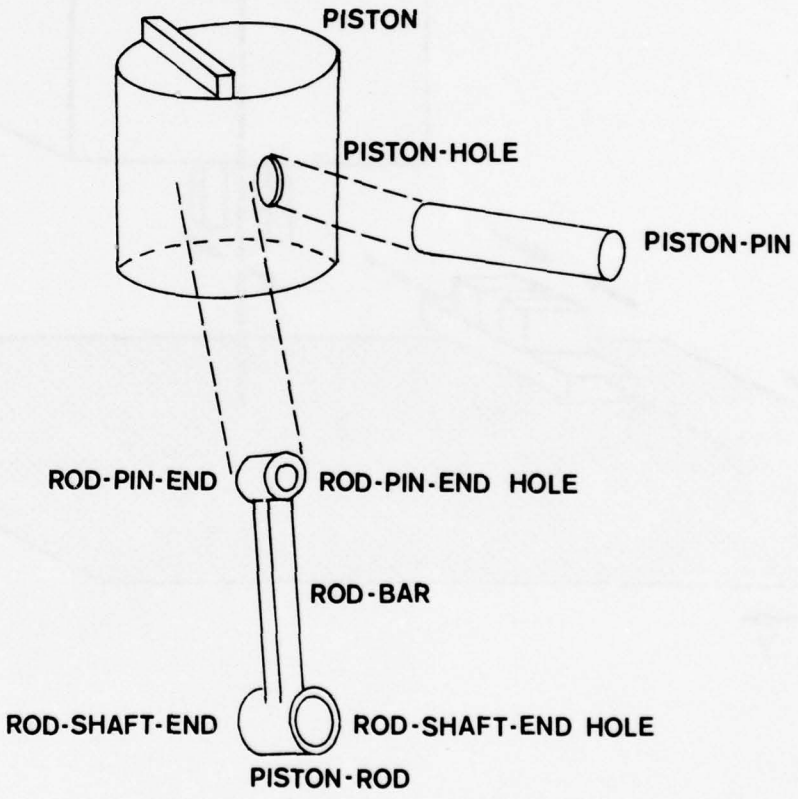


Figure 1.1 - The piston subassembly from the ENYA 15-III model aircraft engine.

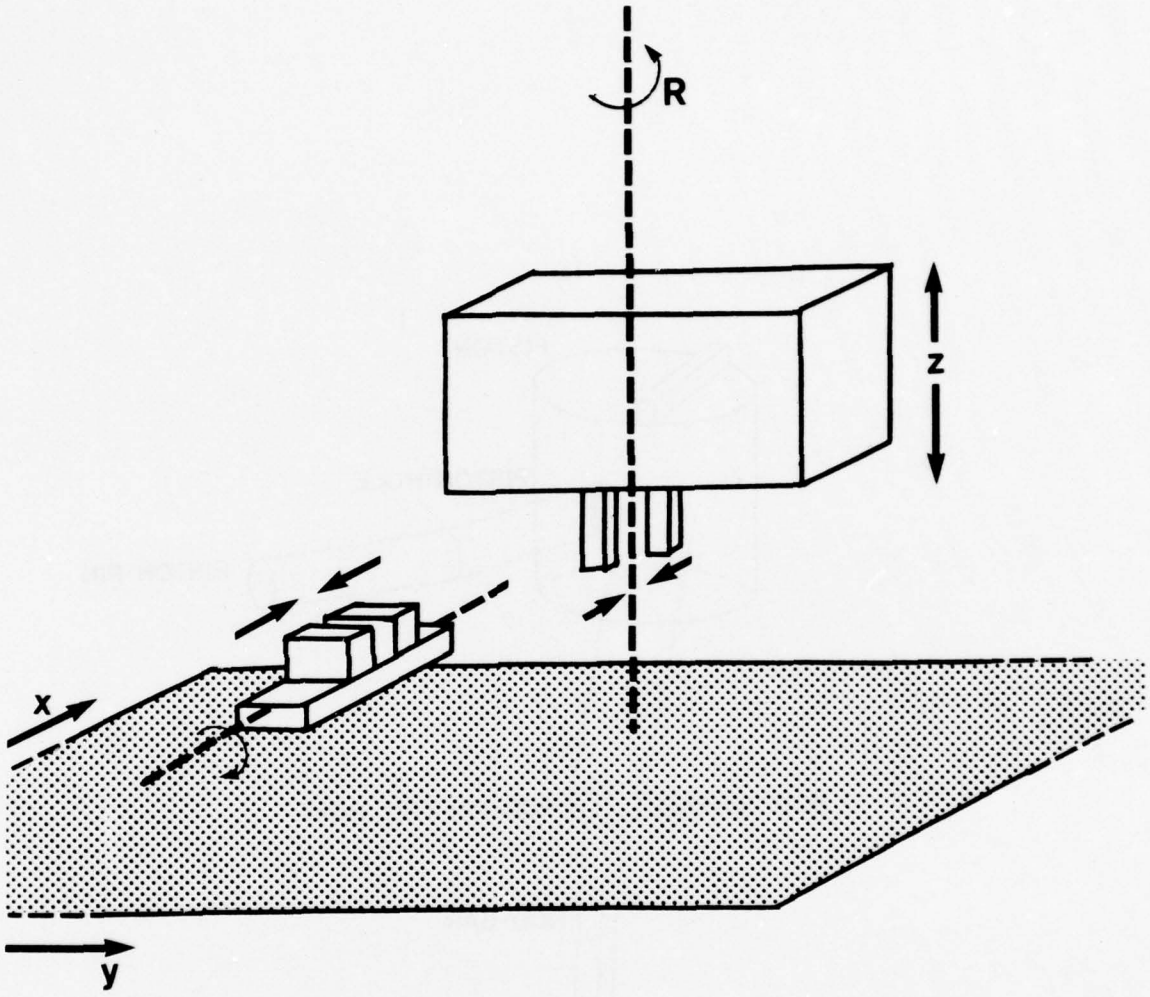


Figure 1.2 - The Little Robot System manipulator configuration. The degrees of freedom are split into (1) an *xy* table, (2) a wrist which can displace and rotate along the *z* axis and (3) a vise which rotates around the *y* axis.

rotates about the *y* axis (see Fig. 1.2). The manipulator is equipped with a force sensing wrist capable of resolving the XYZ components of the forces and torques acting on the wrist. This allows the manipulator to generate and detect forces. The use of force feedback enables the Little Robot System to perform precise assembly tasks whose critical clearances are below its positional accuracy [Inoue].

The first task is to determine a plan for the assembly. A plan is a sequence of operations which will take the parts from their original positions through a series of intermediate states to the final assembly. The hard part of this assembly is inserting the piston-pin through the piston pin-hole and through the piston-rod. The obvious way to do this (for a human) is to line up the holes in the piston-rod and the piston and then push the piston-pin through both holes. This operation is impossible using the manipulator configuration we have described. Recall we only have two sets of parallel fingers available; one set is the hand, the other the vise. This restriction forces us to break the problem up into three parts. First we insert the pin partway into the piston. Then we place the piston-rod's pin-end onto the pin inside the piston and finally, we push the pin through the rod and the piston-hole. This is, of course, far from enough to specify the assembly completely, much less write a program to carry it out. If the reader is anything like the author when first faced with this problem, he will have very little idea of precisely what to do next. I will claim later that it is feasible to develop a computer system that will take this formulation of the solution and generate a program to carry it out. Before we give support for this claim let us consider what remains to be done.

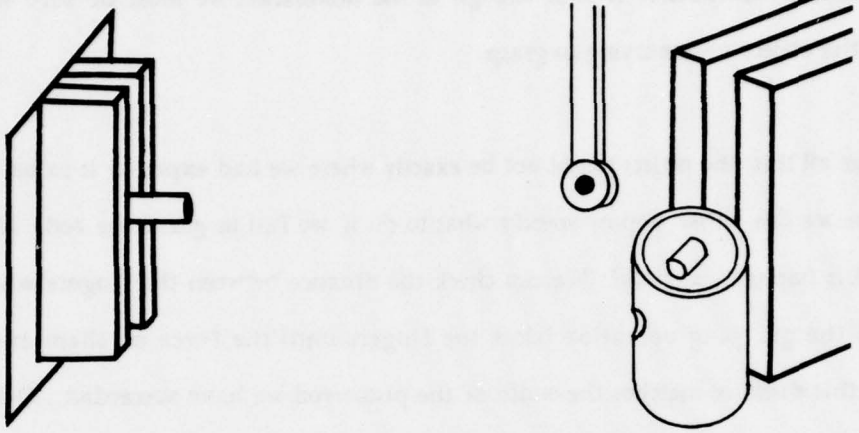
The *description* of the piston assembly given above simply specifies three snapshots of the system state. Each state is specified in terms of the spatial relations between the parts involved. The verbs used in the description (insert, place and push) give some information as to the nature of the operation necessary to achieve each state. Notice that no mention was made of the manipulator. The constraint that only one hand and a vise is available dictates the nature of

the solution, but the manipulator motions necessary to carry out the solution are not specified, nor are they obvious. We still have to decide what to hold in the hand and what to hold in the vise. We also have to settle on the original orientation of the parts. Of course, how to set up the parts depends on what we are going to do with them. Indeed, what are we going to do with them?

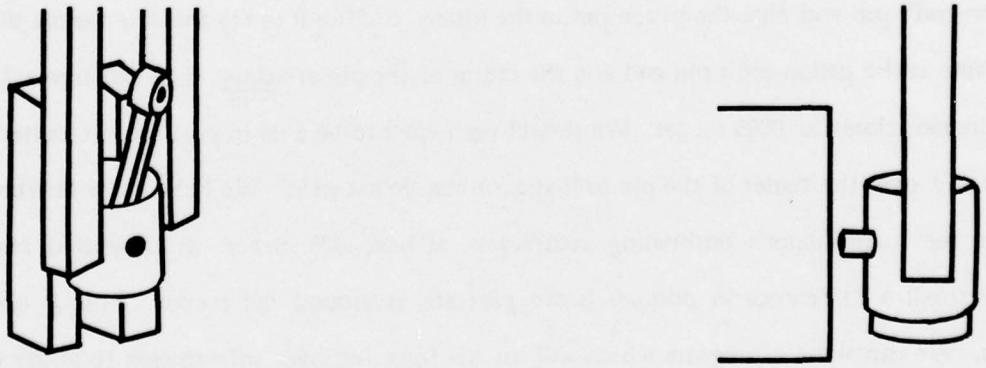
We can put the pin in the vise so that its main axis is horizontal (Fig 1.3a). Then we can slip the piston onto the pin such that the cavity is facing upward. Now we place the rod onto the portion of pin that is projecting into the piston's cavity (Fig 1.3b). We can then grasp the piston, open the vise, pull the piston out, and after closing the vise, push the part of the pin protruding from the piston against the vise (Figs 1.3c&d). We will postpone consideration of how to obtain this type of ***assembly plan*** and instead discuss what remains to be done.

How are we to grasp the parts? To a human assembler this might seem a trivial question but it definitely is not trivial in the context of an assembly program. For each operation we must specify the position and orientation of the fingers such that they can securely grasp the object to be moved. Does this means that we must know the position and orientation of the object? No! It means we must know the position and orientation of the part of the object to be grasped. It is not sufficient to tell the manipulator to grasp the piston-rod located at such and such a location. We must specify precisely, in manipulator coordinates, the location of the particular part of the piston-rod we want to grasp is. We can then move the manipulator there.

Now we can move the manipulator to the position and orientation of the center of the piston-rod's shaft-end. Or, can we? We can try to move the manipulator to where the end of the piston-rod is. Actually, the hand moves as close to the rod as it ever gets to any destination, which, of course, depends on its previous position, among other things. Also, we probably knocked the manipulator against the vise in our hurry to reach the piston-rod. We must specify



a b



c d

Figure 1.3 - Stages in the assembly of the piston:
(a) The pin in the vise
(b) Inserting the piston-rod on the pin held in the vise.
(c) Grasping the piston so as to remove the assembly from the vise
(d) Pushing the pin through the piston hole by pushing it against the closed vise.

a trajectory that is not likely to destroy the manipulator or disturb any of our carefully achieved positions. Once the manipulator is near enough to the destination we must be sure we don't knock against the object we are trying to grasp.

Of course, after all this, the object might not be exactly where we had expected it to be. If this is the case then we can either stop or specify what to do if we fail to grasp the rod. How can we tell when this happens, anyway? We can check the distance between the fingers when they have finished the grasping operation (close the fingers until the force on them exceeds a threshold). If this distance matches the width of the piston-rod we have succeeded. Otherwise, we might try searching nearby. By the way, what was the width of that end of the piston-rod? We better know that if we are to insure that we are not grasping the wrong end. We must also specify the search pattern and what to do if we find the rod but it is rotated with respect to the fingers.

We finally grasped the piston-rod at the desired position. We now have only to place the piston-rod's pin end onto the piston-pin in the piston. Suffice it to say that the radius of the of the hole at the piston-rod's pin end and the radius of the pin are close. The difference is below .001 inches, closer to .0005 inches. We should not expect to be able to position the center of the rod's end over the center of the pin to that accuracy. What next? We have force feedback and while the manipulator's positioning accuracy is, at best, .005 inches; its positional resolution (how small a difference in position it can generate) is around .001 inches. This is cause for hope. We can write a program which will use the force feedback information to guide the rod onto the pin. Happily, somebody already has done this for us. Inoue has developed a program which, driven by force feedback, does peg-in-hole insertion [Inoue].

Presumably, we can now position the rod somewhere near the tip of the pin and call the peg in hole program. Not exactly! Remember that the tip of the pin is inside the piston. Not only do

we have to get the rod's end to be near the tip of the pin but it must also be inside the piston. This is strictly our responsibility since Inoue's program does not know about this sort of thing. There is not much room inside the piston, so merely achieving this preliminary step might be tricky. But let us assume we manage to do it. Can we then just call Inoue's program to do the actual insertion? Not quite. Are we sure that we have met the position and orientation assumptions he makes? For one thing, in our case, the axis of the hole does not pass between the tips of the fingers. Inoue's program assumes that the motions of the hand correspond directly to motions of the hole. We, on the other hand, have decided to hold the rod at the opposite end from the point of insertion. This is because the insertion is happening inside the piston. Inoue's program must be changed a bit.

Like most assembly programs, Inoue's also has parameters. The length of certain motions and the magnitude of some forces are not specifed. We must pick these parameters for the task at hand.

Inoue's program also lacks error detection capabilities. When the manipulator is told to move in a given direction until a force above some threshold is felt, no position bounds are specified. In our situation this is risky because if we bump the rod against the side of the piston one of three things will happen: (1) the piston will turn on the pin, (2) the rod will turn in the hand, (3) the manipulator will go sailing into the blue after having turned the piston and jarred the rod loose.

Inoue's program is not so much a general utility program as a specification of an assembly strategy to be adapted to many different geometries.

I believe this scenario identifies most of the major problems we face in specifying an assembly operation for a computer controlled manipulator. Let us summarize them:

(1) Describe the objects to be assembled.

(2) Specify a plan for the assembly. The details of the plan will depend on the manipulator configuration available and on the capabilities of the assembly system.

(3) Determine the grasping position and orientation for the objects involved in each operation.

(4) Determine a collision free path between the origin and destination of all motions.

(5) Modify assembly strategies to fit the particular geometric environment. The parameters must also be specified.

(6) Examine the strategies for likely failures.

With this list of problems in hand, we can formulate a design for a system that deals with some or all of them. Before we can do this we must organize the problems more crisply. Many approaches are possible. The approach outlined in the next section has proved quite fruitful.

I.2 The Nature of the Approach

We have characterized a Mechanical Assembly System as follows:

A Mechanical Assembly System is a system that transforms an assembly description into a program for a computer controlled manipulator.

This sentence embodies a system design. It isolates two abstract objects and a transformation between them. The two objects are (1) an assembly description and (2) a manipulator program. What precisely is the nature of these objects? Let us consider them in turn.

I.2.1 The Assembly Description and the Assembly Plan

An *Assembly Description* is an ordered list of *assembly operations* or *steps*. Assembly operations are high level commands such as INSERT, PLACE, GRASP and UNGRASP. The positions and orientations of parts, initially or at their destination, may be specified symbolically. The operations need not be fully specified. For example "Insert A into B" might be an assembly operation. Its resulting state has "A FITS-IN B" as a constraint. Notice, however, that there are still many degrees of freedom in the positions of A and B.

Our chief design goal is to allow the initial description, obtained from the user, to be as natural as possible subject to the well-ordered constraint and the use of the available operators and descriptors. The requirement that assembly operations must be ordered within a description rules out as a legal Assembly Description the "ideal" Assembly Description, the one consisting simply of a description of the parts and the assembled object. There are other ways of making an Assembly Description more natural. The following properties allow quite a bit of flexibility in the Assembly Description.

(1) Some of the parameters of the assembly operations may not be specified.

(2) Assembly operations need not have their prerequisites explicitely established in the preceeding assembly steps.

(3) Manipulator motions are not explicitely specified. The vocabulary of assembly operations is not designed for specifying manipulator motions. It is designed for expressing spatial relationships between parts.

Now some important definitions can be made. An Assembly Description that has no instances of (1), and (2) is called a *Complete Assembly Description* or *Assembly Plan*. One that has been expanded to include explicit manipulator commands (3) is a *Manipulator Program*.

I.2.2 The Manipulator Program

The output of a Mechanical Assembly System is a manipulator program. LAMA's target language LLAMA is described in Appendix 1. The important aspect of these programs is that they deal exclusively with positions, orientations and forces; all in the frame of reference of the manipulator. This means that all the degrees of freedom present in the initial Assembly Description must be constrained. What remains is a program to achieve the assembly states, taking into account uncertainties in the position of objects and the errors in the manipulator. To compensate for these errors and uncertainties, feedback strategies must be used. These strategies specify how force and position readings are to be used to guide the manipulator to the desired state.

An excellent example of such a strategy is embodied in Inoue's peg in hole insertion programs [Inoue]. The programs were written for the Little Robot System. They were used to put together a bearing assembly consisting of eight parts with clearances around 0.001 inches. The insertion is divided into three steps: DROP-INTO, MATE and PUSH-INTO. The first operation partially drops a shaft into the hole. The next phase adjusts the relative position and orientation between the two until the axes of the peg and the hole are properly aligned. In the third phase the peg is pushed into the hole smoothly until it arrives at the bottom.

The programs use two clever techniques to insure that the peg lands at the hole. The first is to introduce a deliberate offset into the aproach trajectory of the hand. This eliminates one of the directions of search should the hole be missed. The other technique is to tilt the peg with respect to the hole. This increases the area over which we can get the peg to drop partway into the hole. The mating operation then locates the edges of the hole by sliding until contact is felt. Then the axes are realigned and the insertion completed under force control. We will look at these programs in more detail in Chapter 5.

I.2.3 The Transformation

The major goal of a Mechanical Assembly System is to transform an assembly description into a manipulator program. We will identify three stages of this process. These stages are defined conceptually and do not correspond explicitly to the stages of processing in LAMA.

(1) Completing the Assembly Description. The initial description together with a description of the available assembly operations leads to a complete Assembly Description or *Assembly Plan*. This stage is called *Assembly Planning*.

(2) Converting the assembly plan into a program that assumes ideal position information and positioning accuracy. Given these two assumptions and a complete Assembly Description it is still necessary to specify the manipulator motions that achieve the desired relationships between the parts. We will call this process the *Pick and Place* phase.

(3) Introducing the feedback strategies. Both assumptions in (2) are untenable. The role of *Feedback Planning* is to expand skeleton programs embodying feedback strategies to carry out the assembly operations. The resulting program is the desired *Manipulator Program*.

We will see that both the Assembly Planning and the Pick and Place phases require knowledge of constraints imposed by the feedback strategy to be employed. The system must be arranged to allow these interactions to occur without mixing these conceptually different stages.

The rest of the report focuses on the second and third stages introduced above, the pick and place and feedback planning phases. The discussion of the assembly planning phase will be mostly speculative (cf Chapter 6).

I.2.4 The Users

The goal of this research is to produce a system which makes the generation of programs for automatic assembly more direct. This led to the requirement that the user interface deal with the operations to be performed on the parts rather than with the manipulator motions necessary to achieve them. Transforming these assembly descriptions into a manipulator program still requires knowledge of manipulator strategies. The system design presented above not only tries to isolate the end user from knowledge of these strategies but also to keep the strategies as independent programs. This allows the system's repertoire to grow smoothly since the strategies are not built in to the system.

The system thus provides a niche for the sophisticated user who can develop and define new assembly strategies. A language is provided to specify the strategies and document their desired effect. The system is then able to instantiate these strategies under different work environments.

I.3 An Example

This section expands the example introduced in section I.1. We now consider how the piston assembly would be processed by future versions of LAMA. This is not based on an actual console session. It is a scenario that serves to identify the role of each of the components in the LAMA design. Section I.4 will outline the subset of the system that has been implemented.

The parts to be assembled must first be described to the system. The potential exists for obtaining these descriptions directly, either from engineering drawings or from a Numerical Controlled Machine tape. Several ongoing research projects are directly concerned with easing the process of acquiring descriptions of complex objects [PADL]. We will assume only the

following capabilities:

(1) An interactive graphics system and an object definition capability similar to the one described in [Lavin & Lieberman].

(2) The ability to specify constraints between features of objects (such as faces of polyhedra) symbolically as in [Ambler & Popplestone].

The user must then use the system interactively to define models of the parts. Complex objects can be described by specifying the relative positions of a few kinds of primitive object types. The primitive objects currently available are shown in Fig. 1.4. Combinations of these objects can be defined and then used as primitives. In this way it is possible to build a library of common object models.

Fig. 1.5 shows a schematic description of the models for the parts in the piston assembly. Notice that the parts are arranged hierarchically. This allows a convenient treatment of subparts of objects. Any desired subparts can be represented as nodes in the part model trees. Each node has information regarding the size, type and relative position of the subpart. Fig 1.6 shows a graphical representation of the piston subassembly components. Notice that all the subparts, including the holes, are approximated as rectangular or octagonal right prisms. This provides a uniform internal representation for all the object types. This representation simplifies the definitions of the spatial modelling operations described in Appendix 2. By generalizing to polyhedra we could approximate the desired volumes to any required accuracy.

The next step is that of describing the assembly. Ideally, we would like to specify the assembly process by simply describing the completed assembly. That level of performance is currently beyond our reach. A more realistic goal is that of accepting assembly instructions similar to those given to people. Chapter 6 includes a characterization of what I believe good assembly instructions to be. An example will help illustrate.

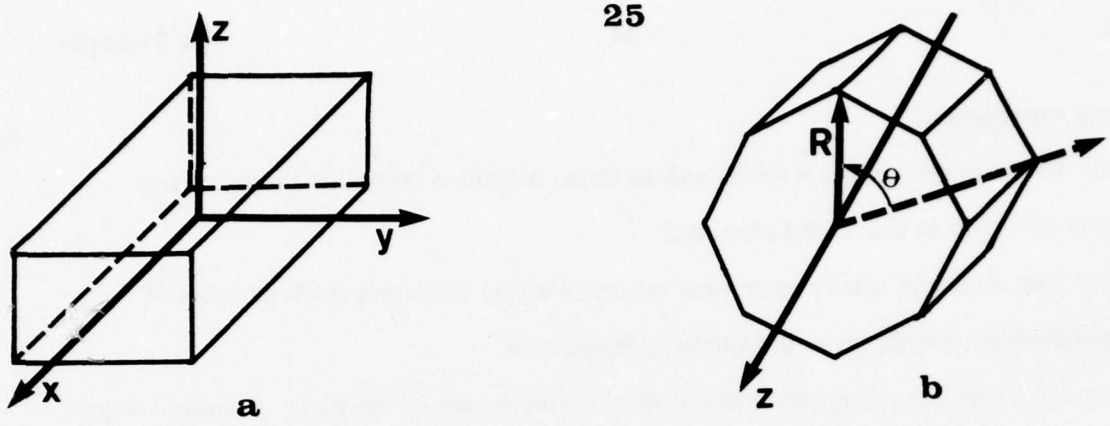
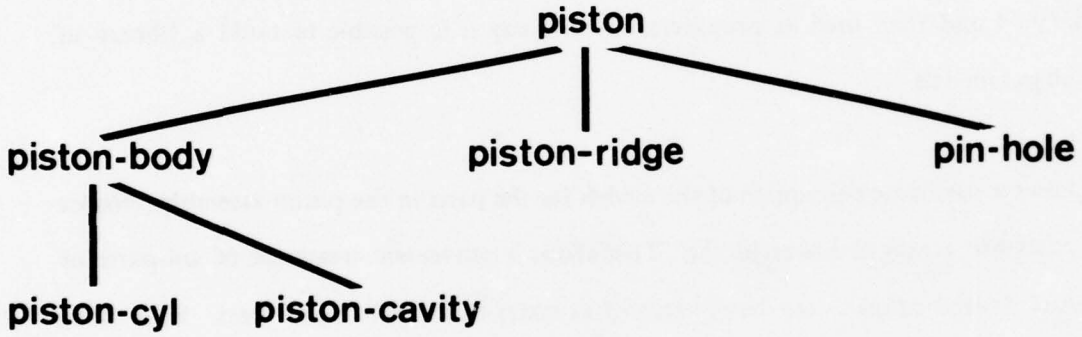


Figure 1.4 - Primitive objects in the modeling system: (a) a cuboid and (b) a polyhedral approximation to a cylinder. Notice that the coordinate system local to each object is located in the center of the objects.



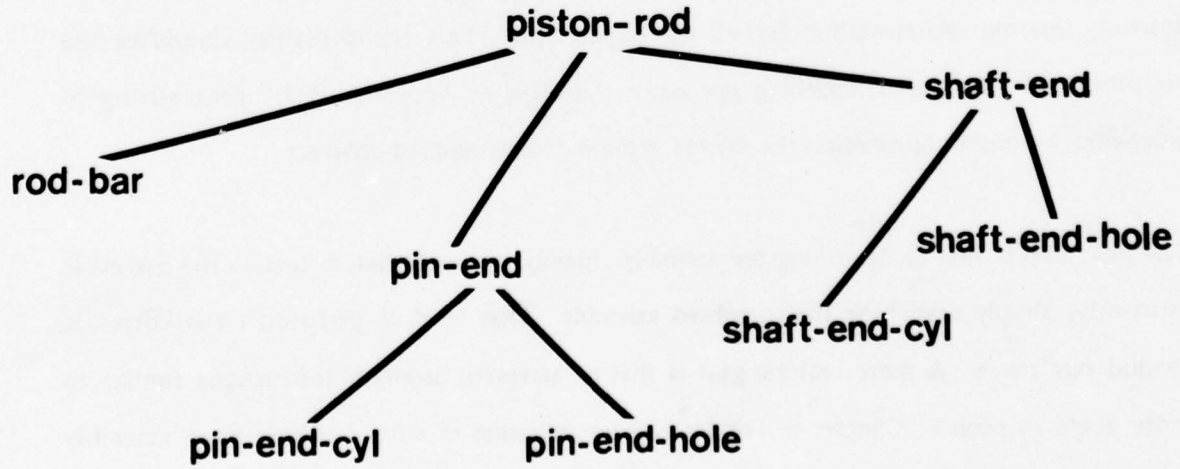


Figure 1.5 - Schematic of parts in piston sub-assembly. This shows the tree-structured relationship between the primitive objects making up the models of two parts of the piston subassembly.

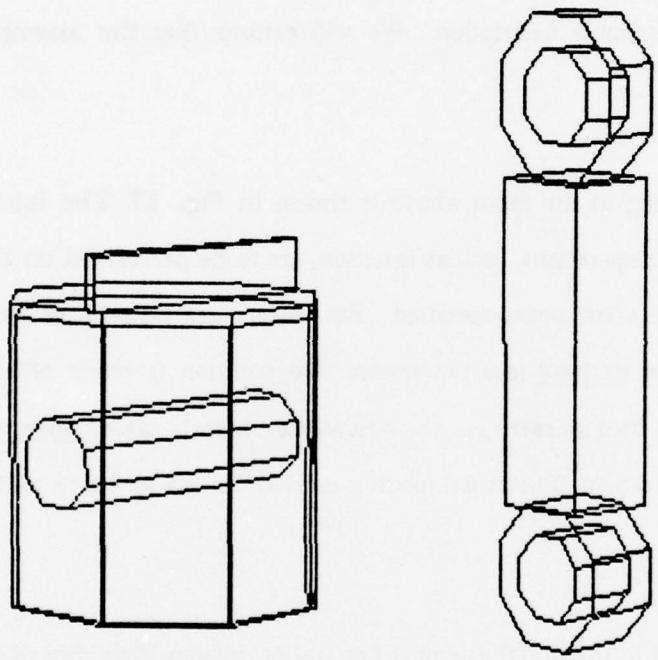


Figure 1.6 - Computer display of piston and piston-rod models. Only a few hidden lines are eliminated in this picture and holes are drawn as solids.

(1) Insert the piston pin partway into the piston.

(2) Place the rod's pin end on the piston pin inside the piston.

(3) Push the pin through the rod and the piston hole.

We would like the user to be able to type his instructions in this format and have the system do the best it can with them. Throughout this report we will ignore the problem of converting an English description into a LAMA assembly description. We will assume that the assembly descriptions themselves are available.

The assembly description corresponding to the input above is shown in Fig. 1.7. The initial assembly description specifies only that operations, such as insertion, are to be performed on the parts. The individual operations are often under-specified. Parameters are missing or only weakly constrained, e.g. insert the pin partway into the piston. No mention is made of the manipulator or of the strategies to be used to carry out the operations. Merely saying insert is not enough to specify an assembly operation. The actual motions carried out are sensitive to the shape and relative sizes of the parts.

Not only are the operations mentioned in the initial assembly description incompletely described but the assembly description as a whole leaves many operations unspecified altogether. In the assembly description shown above most of the prerequisite conditions for the individual assembly steps have not been achieved by the time the step is to be performed. For example, inserting the pin in the piston reqires one of the parts to be in the hand and the other to be securely fastened, either in the vise or a jig. There is no mention of any of this in the initial description.

The next step in the transformation from user input to manipulator program is to completely specify the assembly description. This is the task of the Assembly Planner. It must first

```
(INSERT OBJ1: [PISTON-PIN]
        OBJ2: [PISTON PIN-HOLE]
        SUCH-THAT: (PARTLY (FITS-IN OBJ1 OBJ2)))

(INSERT OBJ1: [PISTON-PIN]
        OBJ2: [ROD SMALL-END-HOLE])

(PUSH-INTO OBJ1: [PISTON-PIN]
           OBJ2: (AND [PISTON PIN-HOLE] [ROD SMALL-END]))
```

Figure 1.7 - Initial Assembly Description for the piston assembly.

```
(GRASP OBJ: [PISTON-PIN])

(PLACE-IN-VISE OBJ: [PISTON-PIN])

(UNGRASP OBJ: [PISTON-PIN])

(GRASP OBJ: [PISTON]
       SUCH-THAT (FACING+ ([PISTON] TOP) DOWN))

(INSERT OBJ1: [PISTON-PIN]
        OBJ2: [PISTON PIN-HOLE]
        SUCH-THAT: (PARTLY (FITS-IN OBJ1 OBJ2) 0.25))

(UNGRASP OBJ: [PISTON])

(GRASP OBJ: [ROD]
       SUCH-THAT: (FACING+ ([ROD-BAR] TOP) UP))

(INSERT OBJ1: [PISTON-PIN]
        OBJ2: [ROD SMALL-END-HOLE])

(UNGRASP OBJ: [ROD])

(GRASP OBJ: [PISTON])

(REMOVE-FROM-VISE OBJ: [PISTON])

(PUSH-INTO OBJ: [PISTON-PIN]
           SUCH-THAT: (AND (FITS-IN [PISTON-PIN] [PISTON PIN-HOLE])
                           (FITS-IN [PISTON-PIN] [ROD SMALL-END])))

(UNGRASP OBJ: [PISTON])
```

Figure 1.8 - The Assembly Plan that would be generated by the Assembly Planner. This is the input to the Feedback Planner.

introduce into the description those operations that will achieve the prerequisites of the operations in the initial description. This requires specification of some high level manipulator commands such as GRASP, UNGRASP and PLACE. Then the operations must be completely specified and strategy choices made for them. The end result of this process is an assembly plan. In this plan each operation is fully specified and the positions and orientations of the parts involved are well constrained. An important point to notice is that the plan still does not determine the manipulator motions necessary to carry out the assembly. The assembly plan corresponding to the assembly description in Fig. 1.7 is shown in Fig. 1.8.

The Assembly Planner operates on (1) models of the parts, (2) a manipulator model, describing the shape and capabilities of the manipulator, (3) an assembly description and (4) the descriptions of the assembly operators. The knowledge of the assembly operators in the Assembly Planner is not at the level of the manipulator motions needed to carry them out. The knowledge is limited to the following:

(1) Options for performing the operation, e.g. a pin can be inserted in a hole either while holding the pin or the object with the hole.

(2) Requirements on the degrees of freedom of motion on the parts involved, e.g. clamped, free to rotate, etc.

(3) What kinds of uncertainties in position and orientation will the operation tolerate, e.g. an insertion will tolerate more uncertainty in the position of a peg in the hand than in its orientation.

(4) Constraints on clearances, sizes, etc. Some insertion strategies will work on a loose fit insertion while others work well in tight fit situations [Inoue].

(5) Constraints on the positions and motions of the resulting assembly.

The Assembly Planner uses this information to help specify the plan. These constraints on the operation seldom determine the plan completely. Added constraints come from: (1) limitations in

the position and orientation capabilities of the manipulator, (2) grasping and collision avoidance considerations, (3) considerations of stability and support. None of these constraints depend on knowledge of the manipulator motions needed to carry out the operation. If they are not enough to completely determine the plan, information about the manipulator commands used to implement the strategy must be used. The Assembly Planner does not examine the assembly strategies directly; rather, it uses the Feedback Planner to evaluate the strategies. The alternative ways of performing an operation are expanded and the number of likely errors compared.

Once the assembly plan has been fully specified, a detailed pick and place computation can be carried out. This will determine precisely where the objects are to be grasped and what paths they must follow to avoid collisions. Both of these operations were performed qualitatively during the assembly planning phase. Now it is done in more detail. Unfortunately, the Pick and Place computation is not independent of the nature of the assembly strategies. Where the object is grasped and where it is placed prior to an operation depends on the details of the operation. The solution is to do the grasp computation at the initial position of the object to be moved, before the operation is instantiated. This determines the range of possible grasp points. After this, the assembly step is expanded. The instantiation process places additional constraints on both the initial position of the part and its grasp point. Then, an exact grasp point is chosen and the path computed after the operation has been expanded.

The pick and place computation exercises most of the spatial expertise of the system. The basic operation in both grasping and collision avoidance is detecting the possibility of a collision by intersecting volumes. In finding a collision-free trajectory we are interested in whether the volume swept out by the manipulator and the object it carries, collides with other objects in the workspace. Similarly, in grasping we are interested in the locations on the object where the hand can be placed such that no collisions will result. Since there are a whole range of grasping

positions for a given object, this amounts to intersecting the volume of the hand, swept out over the possible grasping positions, with the workspace.

We have characterized the types of grasping positions for the primitive objects as a series of *grasp sets*. Grasp sets are parameterized ranges of hand positions over a surface of the object. Fig. 1.9 shows a graphical representation of the grasp sets for cuboids and cylinders. Complex objects are analyzed by considering how to grasp each of their component objects while taking into account the interactions with other parts of the object as well as with the environment.

Knowledge of the details (at the manipulator level) of the assembly operation is necessary to fully specify the assembly plan. The insert operation requires that one of the objects be fastened so as to counteract the forces generated by the insertion process. What precisely are these forces? Clearly this depends on the details of the insertion method. How are these details determined? Fig. 1.10 shows the system's representation for the *peg-in-hole insertion* strategy. It is very similar to the program presented in [Inoue]. We will postpone the details of syntax until Chapter 5. The Feedback Planner can simulate this *skeleton program*, predict contacts and estimate the direction and magnitude of the forces that will be produced.

Notice that each step in the skeleton program is annotated by the constraints it generates between the manipulated parts. This information can be used in two ways:

(1) To generate numerical values for parameters in the programs. For example, the size of the shift in the y direction in the DROP-INTO operation can be determined by examining the constraints it is meant to achieve.

(2) To generate tests for likely failure situations given the particular execution environment. A good example of this is the operation of moving the piston-rod near the piston-pin for the insertion of the rod's pin-end onto the pin inside the

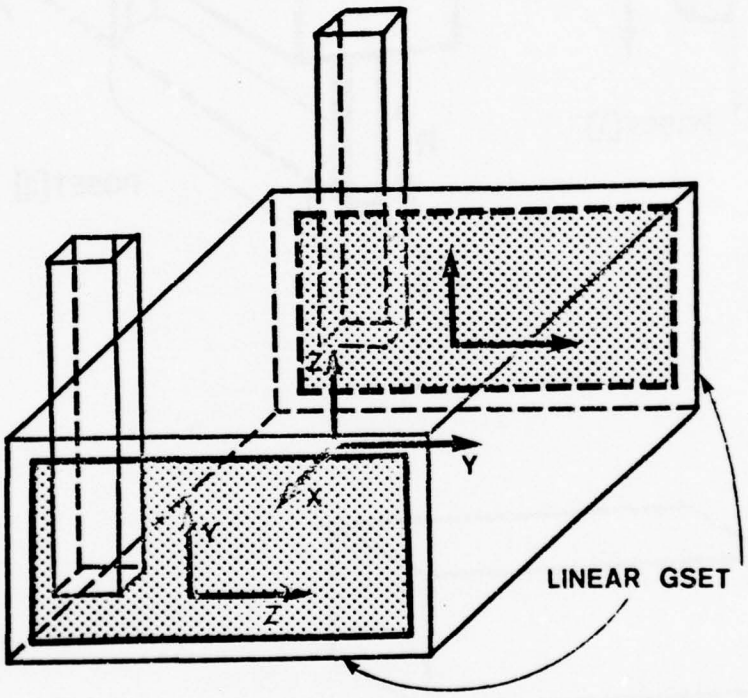


Figure 1.9 - Grasp Sets. This shows the four basic types of grasp sets PGSET[1], PGSET[2], PGSET[3] and LGSET.

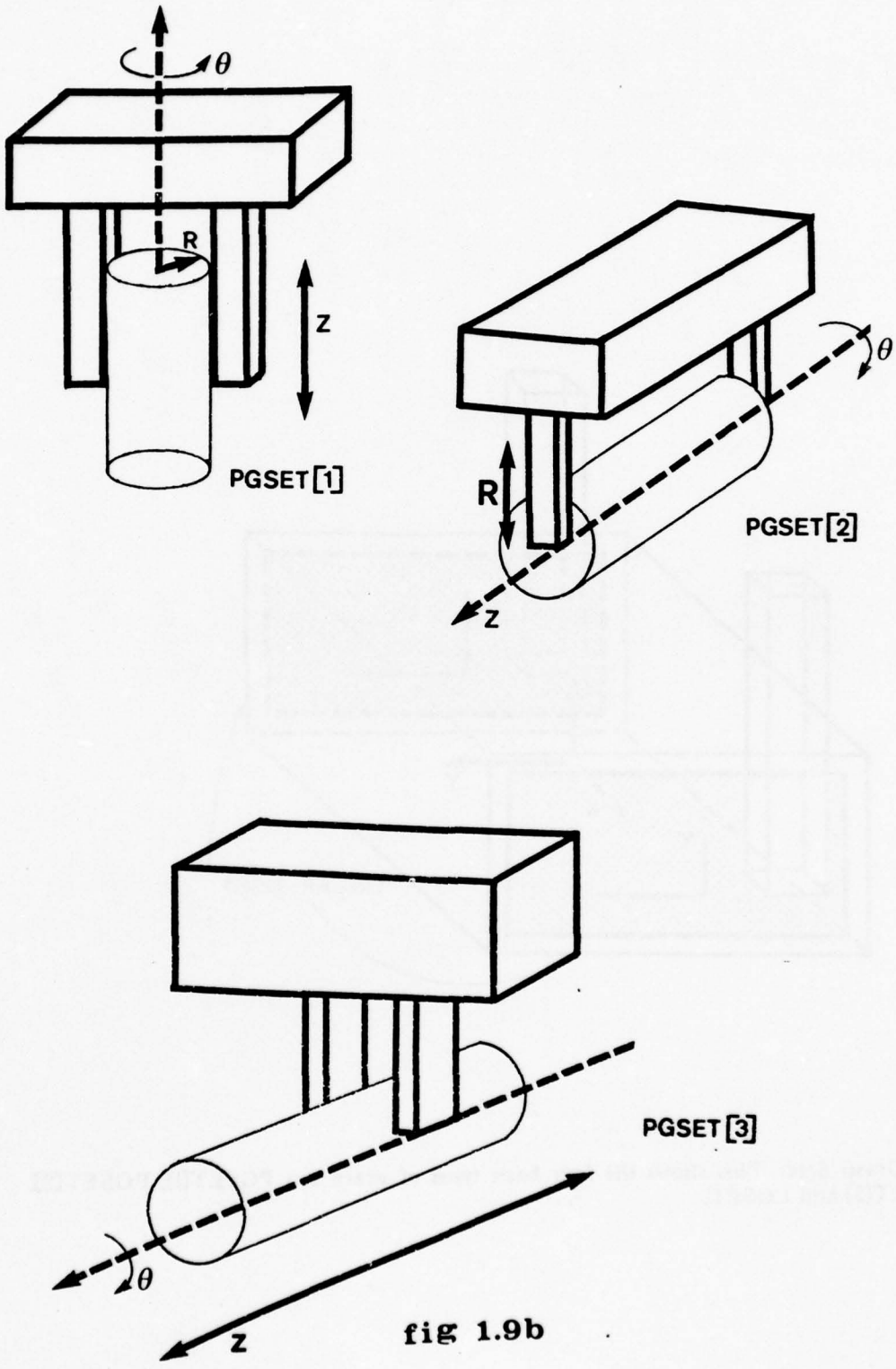


fig 1.9b

```
(STRATEGY PEG-IN-HOLE (PEG HOLE)
          (TYPE (PEG CYL) (HOLE CYL-HOLE))
          (REFERENCE (ALIGNED&CENTERED (REFERENCE X) (HOLE FRONT)))
          (PRE-REQS (CLEARANCE < 0.01))
          (INITIAL (AND (ALIGNED&CENTERED (PEG FRONT) (HOLE-FRONT))
                        (IN-FRONT-OF PEG HOLE)))
          (DROP : (DROP-INTO PEG HOLE)
           SUCH-THAT (PARTLY (FITS-IN PEG HOLE)))
          (MATE : (MATE PEG HOLE)
           SUCH-THAT (ALIGNED- PEG HOLE))
          (INSERT : (PUSH-INTO PEG HOLE)
           SUCH-THAT (FITS-IN PEG HOLE)))

(STRATEGY DROP-INTO (PEG HOLE)
          (ROTATE : (CHANGE R BY (RADIAN 0.1))
           SUCH-THAT (ALMOST (ALIGNED- PEG HOLE) (RADIANS 0.1)))
          (SHIFT : (CHANGE Y)
           SUCH-THAT (LEFT-OF (PEG CENTER) (HOLE CENTER)))
          (LANDING : (CHANGE X)
           SUCH-THAT (CONTACT (PEG FRONT) (HOLE FRONT))))

(STRATEGY MATE (PEG HOLE)
          (EDGE+ : (CHANGE Z)
           SUCH-THAT (AND (ABOVE (PEG CENTER) (HOLE CENTER))
                          (CONTACT PEG (HOLE SIDE))))
          (SAVE1 : (SETQ Z1 ZPOS))
          (EDGE- : (CHANGE Z)
           SUCH-THAT (AND (BELOW (PEG CENTER) (HOLE CENTER))
                          (CONTACT PEG (HOLE SIDE))))
          (SAVE2 : (SETQ Z2 ZPOS))
          (CENTER : (MOVE Z)
           SUCH-THAT (BETWEEN (PEG CENTER) Z1 Z2))
          (CONTACT : (CHANGE Y)
           SUCH-THAT (CONTACT PEG (HOLE SIDE)))
          (MATE : (CHANGE R WITH (AND (ZFORCE = 0.) (YFORCE = "MAINTAIN-CONTACT")))
           SUCH-THAT (ALIGNED- PEG HOLE)))

(STRATEGY PUSH-INTO (PEG HOLE)
          (PUSH : (CHANGE X WITH (AND (YFORCE = 0.) (ZFORCE = 0.)))
           SUCH-THAT (FITS-IN PEG HOLE)))
```

Figure 1.10 - Peg-in-hole strategy. The representation of Inoue's peg-in-hole insertion strategy in LAMA.

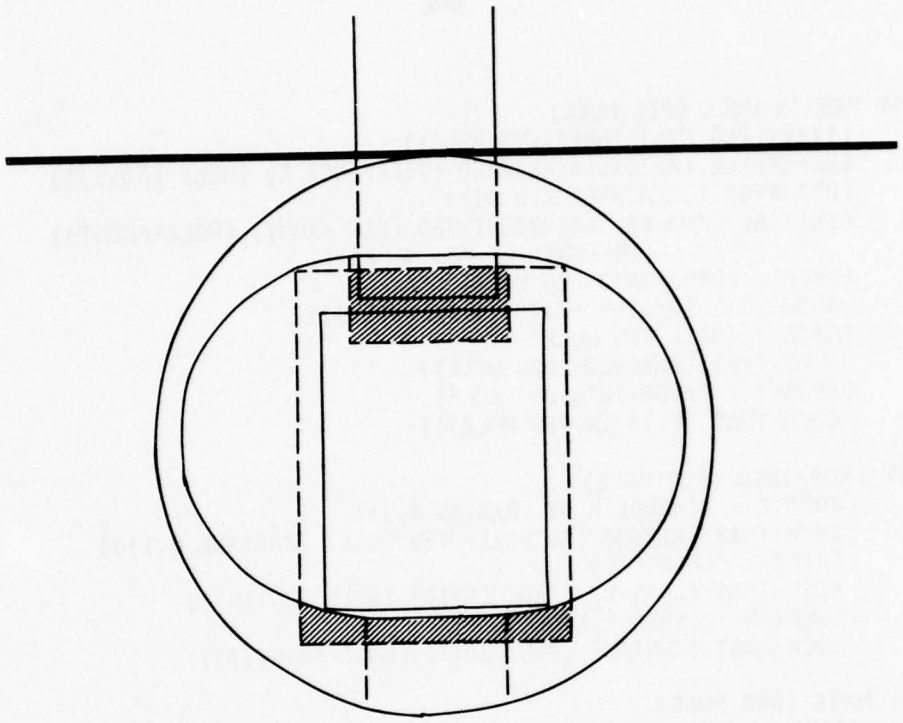


Figure 1.11 - Potential contacts generated when the piston-rod is placed in front of the piston-pin inside the piston are indicated by the shaded area. This is a top view.

piston. By examining the clearance between the tip of the pin and the piston wall given the errors in grasping and positioning, we can predict that sometimes the rod will contact either the pin or the piston (see Fig 1.11). A test for this situation can be generated and instruction as to corrective action could be solicited from the user.

The result of the operation of the Feedback Planner is a manipulator program.

I.4 The Rest of the Report

The system outline presented in the last section provides the outline of the report.

The research reported here has focused on the process of transforming the assembly plan into a manipulator program. The assembly plan will be the input to the system. The first operation will be the pick and place computation. Once the grasp points and trajectories are computed, the Feedback Planner simulates each step of the plan independently. The simulation determines the parameters in the skeleton programs and detects likely errors. The result of this operation is the manipulator program.

Chapter 2 deals with parts description and representation. This chapter describes the mechanisms actually implemented. They are rather crude since this research has not focused on this problem. Much work on this problem has been done elsewhere.

Chapter 3 considers position and orientation constraints. This chapter presents a rather simple scheme for representing geometric constraints. These constraints are crucial to the description of assemblies and the representation of assembly strategies. The current implementation of the scheme is described.

Chapter 4 discusses the details of the pick and place computation. This is a hard chapter. It introduces most of the spatial mechanisms needed throughout the system and thus lies at the core of the report. It should at least be skimmed. The implementation of the method is described.

Chapter 5 presents the design and discusses a partial implementation of the Feedback Planner. This is an important chapter. The techniques are similar to those seen in Chapter 4 but they are applied to some very challenging problems. The implementation of the ideas in this chapter is not complete.

Chapter 6 examines the role of the Assembly Planner in more detail. This discussion is speculative since no implementation work was done on this phase of the system.

Chapter 7 presents a list of problems for further research and very briefly summarizes the conclusions.

Appendix 1 describes the target language of the system. It is called LLAMA (Low - level Language for Automatic Mechanical Assembly).

Appendix 2 describes in detail the spatial modeling capabilities of the system.

The main goal of the research reported in this report was to explore the domain of mechanical assembly to a sufficient depth to isolate the problems involved in a Mechanical Assembly System. This analysis led to a proposal of how to subdivide the problems for further study. The implementation of selected parts of the system served to test the feasibility of the design. The implemetation should be viewed not as the final solution to the problems isolated in the research but as a path toward the definition of the problems and their relationships. Wherever

possible I have tried to point out the crucial features of the implementation.

I.5 Relation to Other Work

This research touches on many subareas in Artificial Intelligence research e.g. object description, modeling three dimensions, planning, etc. It is also part of the recent trend towards applications of advanced computer technology to productivity technology e.g. mechanical assembly, parts inspection, electronic repair, etc. [Winston]. This section will be limited to a brief review of other work that is directly relevant to automating mechanical assembly.

There are, at least, five projects that have direct relevance to the task at hand. These projects are being conducted by IBM Thomas J. Watson Research Center, Stanford AI Laboratory, Stanford Research Institute, Univ. of Edinburgh and C. S. Draper Labortory. These systems have different goals and methodologies.

The IBM system design, AUTOPASS [Lieberman & Wesley], is closest to the LAMA design. It is to be imbedded in PL/I and will provide the user with a selection of high level assembly operations, the most general being a PLACE command in which the destination is specified as geometric relations between objects.

The Stanford system, AL [Finkel et. al.], is a complex Algol-like language with many new data-structure and control primitives. The design includes a Very High Level Language capability. Both AL and AUTOPASS, as well as LAMA, rely to a large extent on modeling the effect on the world of the assembly operations.

Russell Taylor in his dissertation [Taylor] develops mechanisms to predict errors in location values from the AL planning model and uses this information to generate AL code

automatically. He also introduces skeleton programs or strategies which describe and summarize the coding decisions that have to be made. The semantics for the strategies are fixed at system creation time.

The Edinburgh group [Ambler et. al.] has focused on the problem of visually locating parts so that they can be pulled out of a heap. They also developed an elegant method [Ambler & Popplestone] for computing the position and orientation of objects given constraints such as AGAINST and FITS-IN. Both are part of a project aimed at a high level mechanical assembly language. I recently learned of an early speculative paper from their group [Popplestone] that anticipated many of the ideas and approaches adopted in this thesis even to the choice of a model aircraft engine as the example.

The goal of all these systems is to expand a task-level description into an program for a specific manipulator. LAMA shares many of the ideas and the approaches of both AL and AUTOPASS. LAMA differs mainly in that it allows user-defined assembly strategies to be manipulated by the system. The key idea is to allow the specification of strategies to be independent from the operations performed by the system.

The Draper Lab [Nevins] group has focused on direct applicability of a mechanical assembly system in the short range. This has led to emphasis on the type of capabilities that can be made available on a minicomputer. They have also carried out extensive theoretical analysis of the requirements of assemblies in terms of manipulator design and control as well as assembly strategies. The Draper group has expressed doubts about the type of strategies that have been used in the Stanford work and that will be used here. They label this type of strategy which focuses on predicting, detecting and correcting errors as "logic branching strategies". They believe is that this type of strategy "may degenerate into searches and thus be slow in execution". They also report they have identified an alternative type of strategy which they call

"information" strategies.

Work being pursued at the Stanford Research Institute on Advanced Automation [Rosen et al] has taken a direction similar to that taken by the Draper Lab. SRI has focused on mechanical assembly techniques with industrial potential in the short range. They also have significant commitments to the industrial applications of computer vision techniques.

Other work, directly related to various subsections of the thesis, will be briefly surveyed when relevant.

II. Object Models

This chapter discusses the mechanisms for describing and representing parts.

II.1 Approaches to Object Modeling

The problem of building computer models of geometrically complex objects has been addressed in the context of graphics [Braid] [Baumgart], computer aided design [PADL] and mechanical assembly [Lavin & Lieberman]. Two major methods have been devised. One method describes the surfaces of the objects by specifying the vertices and edges of the faces of polyhedra or, for curved objects, the crossections or surface patches. I will call this the *surfaces* approach [Appel]. The other method is to approximate complex objects by composing several simpler volumes. This method I will call the *solids* approach [Braid] and [Grossman]. There also exist some hybrid systems that allow both types of descriptions [Baumgart].

The surface approach, although very general, is cumbersome for the user. On input, it requires the specification of the surfaces as if they were independent of each other. The resulting representation is difficult to use for any purpose except displaying the appearance of the object. The solids method, on the other hand, is less general but often reduces the burden on the user. A large class of objects can be simply described as combinations of a few simple solids. The representation also lends itself to the kind of operations performed by an assembly system. This claim can only be substantiated much later when these operations are described.

The solids technique depends on a method for specifying the positions and orientations of the component solids. The most common method is to specify the coordinate transformations

between coordinate systems based in the primitive objects. This method is simple and powerful and leads to efficient implementations. Its only drawback is that the position of an object in a simple relationship to another, such as two faces touching, is almost as difficult to specify as an arbitrary position and orientation. Users find this characteristic annoying. Augmenting the descriptive repertoire to include simple geometrical *constraints* would, I believe, noticeably simplify the description process. Chapter 3 discusses some methods for representing symbolic position and orientation constraints.

II.2 The Representation of Objects

Objects are represented internally as LISP atoms with complicated property lists. The property list contains the shape, position and other information specific to the object as well as information on its relationship to other objects. Mechanisms are available for associating demon procedures with particular properties of object types. If-added and if-removed methods can be activated by the processes of insertion and deletion. These demons are used to keep the data base consistent.

Complex objects are approximated by combining simple objects hierarchically. This combination process defines a tree. Each of the nodes of the tree can be treated individually as an object. The leaves of the trees are instantiations of a few primitive object types (Fig. 1.5).

II.2.1 Primitive Objects

In the current implementation the primitive objects are polyhedral solids whose crossections are regular polygons. In fact, only two primitives are used: (1) a rectangular solid (cuboid) and (2) an octagonal solid which is meant to approximate a cylinder. Generalizing the available primitives to arbitrary polyhedra would not be a major undertaking but, I believe, would add

little to the understanding of the basic issues explored in this report.

Each primitive object has properties that specify its size parameters, vertex points, equations for the planes of the faces, generalized position and orientation, etc. Details are described in [Fahlman].

Generalized position and orientation of objects (position and rotations) are represented by 4x3 arrays called AT arrays [Fahlman]. They are equivalent to the homogeneous coordinate arrays popularized in [Roberts]. The rightmost column, representing the scaling transformation, is omitted. Only the 3x3 rotation matrix and the (x y z) displacement are kept. Each AT array represents a coordinate transformation generated by three angles (Euler angles) and a displacement. We associate with each object a local coordinate system represented by a base coordinate system and an AT array specifying the coordinate transformation from the base. All objects store the AT array representing the transformation from the manipulator's frame of reference to the object's local coordinate system. Optionally, the AT array relative to another object's coordinate system can be stored, as well.

The fourth row of an AT array holds the vector displacement. The first three rows hold the rotation matrix. It is useful to note that the first row is the unit vector cooresponding to the x axis of the new coordinate system specified by the transformation, relative to the base. The second row is the y axis and the third is the z axis. Of course, these three vectors are mutually perpendicular, i.e. their pairwise dot products are equal to zero and the cross product of any two rows yields the third. These facts will prove useful later (cf Chapter 3).

II.2.2 Complex Objects

Complex objects are represented as unions of other objects, simple or complex. The primitive objects are allowed to have negative volumes. In this way holes and cut-outs can be treated uniformly as objects. The cavity of the piston which was shown in Chapter 1 as a simple cylindrical hole is actually represented as two cylindrical holes and a cuboid to approximate its elliptical crossection (Fig. 2.1).

Each complex object has an optional AT array indicating a reference frame for the entire object. For example, a model of the manipulator hand can be built up out of a large cuboid, representing the wrist, and two smaller cuboids representing the fingers. The complex object representing the hand can then have a coordinate system centered between the finger tips.

I have not implemented a scheme that allows the use of intersections of primitive volumes as elements of complex volumes. This would be necessary to obtain a more general object definition facility.

II.3 The Piston Rod: An example

To illustrate the use of these mechanisms, this section presents the model of the piston rod for the model aircraft engine. The first step is to create the component parts of the object. In this case the components are the piston-rod's shaft, the pin-end (and the pin-end's hole) and the shaft-end (with its corresponding hole).

```
(OBJECT (TYPE RECT) (X 0.2) (Y 0.2) (Z 0.62) (NAME BAR))
(OBJECT (TYPE CYL) (RADIUS 0.156) (LENGTH 0.114) (NAME SHAFT-END-CYL))
(OBJECT (TYPE CYL-HOLE) (RADIUS 0.089) (LENGTH 0.114) (NAME SHAFT-END-HOLE))
```

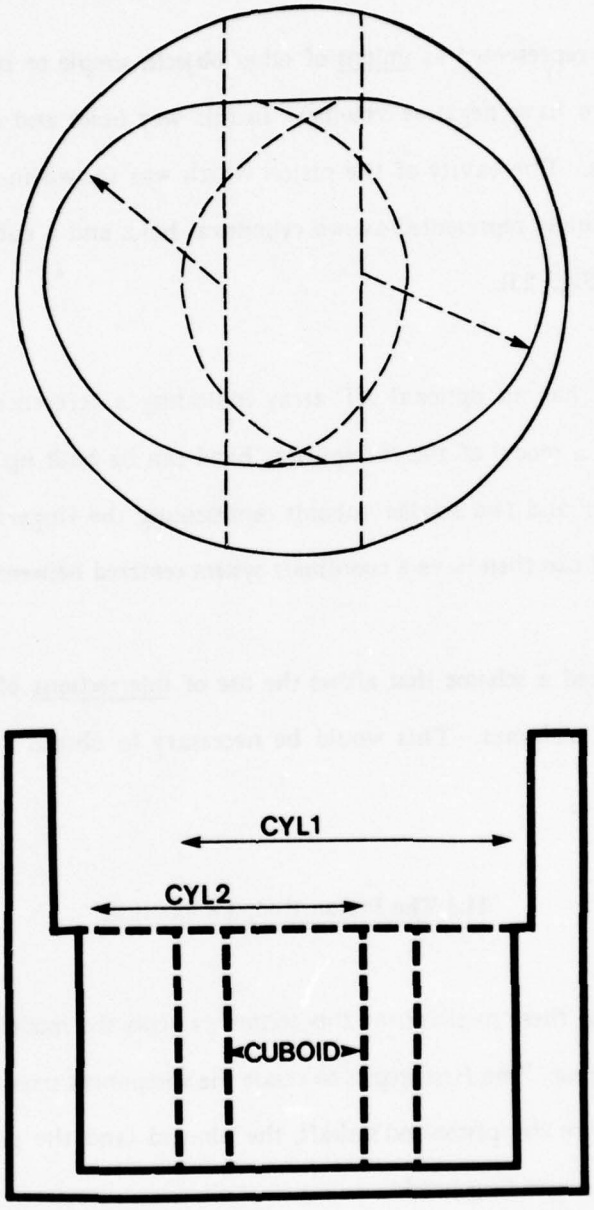


Figure 2.1 - Complex piston cavity model. This shows how to approximate a cavity with elliptical crossection for the piston model using one rectangular and two cylindrical volumes.

```
(OBJECT (TYPE CYL) (RADIUS 0.134) (LENGTH 0.16) (NAME PIN-END-CYL))
(OBJECT (TYPE CYL-HOLE) (RADIUS 0.081) (LENGTH 0.16) (NAME PIN-END-HOLE))
```

The next step is to indicate the relationships between the various parts. The LINK operation indicates the coordinate transformation between the local coordinate systems of two objects. The relationship is enforced by demon processes that update both objects when the position or orientation of either one of the objects changes. The simplest links in the model of the piston rod are the relationships of the holes to their corresponding cylinders since they are aligned and concentric.

```
(LINK SHAFT-END-HOLE SHAFT-END-CYL)
(LINK PIN-END-HOLE PIN-END-CYL)
```

We then have to place the hollow cylinders at the ends of the bar. Since we already have linked the holes to the cylinders we can treat each pair as a unit in the linking operation. The offset in the z direction is the sum of half the length of the bar and the radius of the cylinder. The angles specify that the cylinders are rotated such that their *z* axes are perpendicular to the bar's *z* axis.

```
(LINK SHAFT-END-CYL BAR (OFFSET 0.0 0.0 0.466)(ANGLES 0.0 π/2 0.0))
(LINK PIN-END-CYL BAR (OFFSET 0.0 0.0 -0.444)(ANGLES 0.0 π/2 0.0))
```

It is this last type of relationship that can be made more transparent by the use of geometric constraints.

```
(AGAINST (SHAFT-END-CYL SIDE) (BAR TOP))
(ALIGNED (SHAFT-END-CYL TOP) (BAR FRONT))
```

The first of these constraints indicates that the z axes of the two objects are perpendicular and specifies the distance from their centers. The second constraint indicates that the z axis of the cylinder points along the direction of the y axis of the bar. This will be further examined in Chapter 3.

III. Position and Orientation Constraints

This chapter deals with the mechanisms available in LAMA for representing position and orientation information. These mechanisms play an important role in several aspects of the system design. They are used (1) in the parts description process, (2) to specify the positions and orientations of objects before and after each assembly operation, (3) to indicate the prerequisites of assembly operations and (4) to describe the desired result of the operations in the assembly strategies. These uses call for representing not only single positions and orientations for some objects but also *ranges* for objects whose position and orientation are constrained but not completely determined. We will judge the methods on ease of use, generality and simplicity. The term position will sometimes be used to refer to both position and orientation.

Figure 3.1a shows a simple block configuration. We can specify the positions of the blocks completely in one of three ways: (1) *absolute*, (2) *relative* and (3) *symbolic*. The absolute positions of the blocks are given by the AT array of each object relative to some known coordinate system such as the manipulator's. The relative mode represents the positions by specifying the AT arrays of objects relative to the coordinate systems of other objects. Specifying the absolute position of any object then determines the position of all objects related to it. The symbolic method specifies symbolic constraints on the positions of the objects and then convert these into relative or absolute positions.

In the scene shown in Fig 3.1a, it is easy for a person to indicate the absolute positions and orientations of each of the component objects. Consider, though, the scene shown in Fig 3.1b. The specification of these positions is more difficult even though the two scenes are simple rotations of each other. This example suggests how awkward absolute positions and

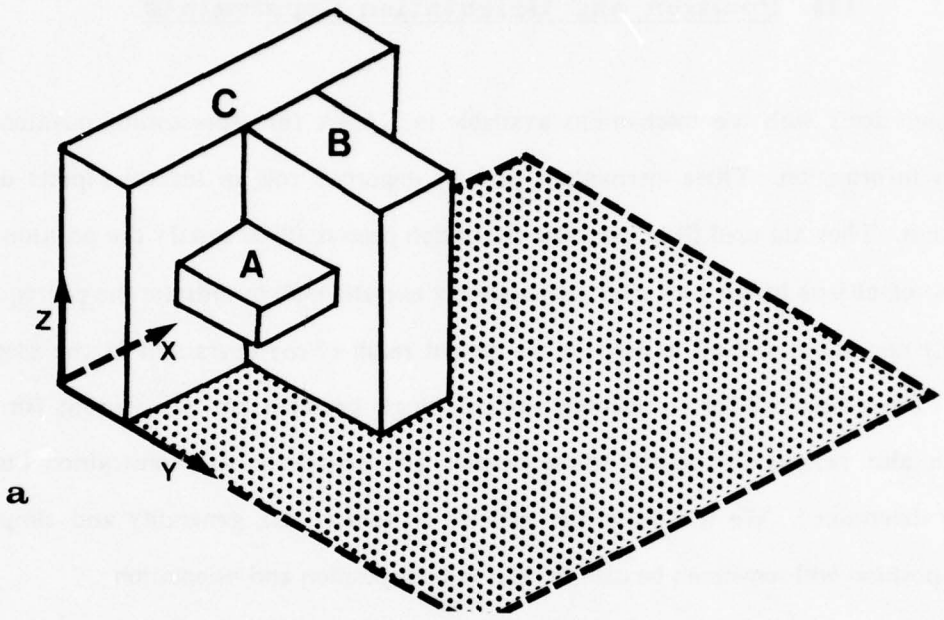
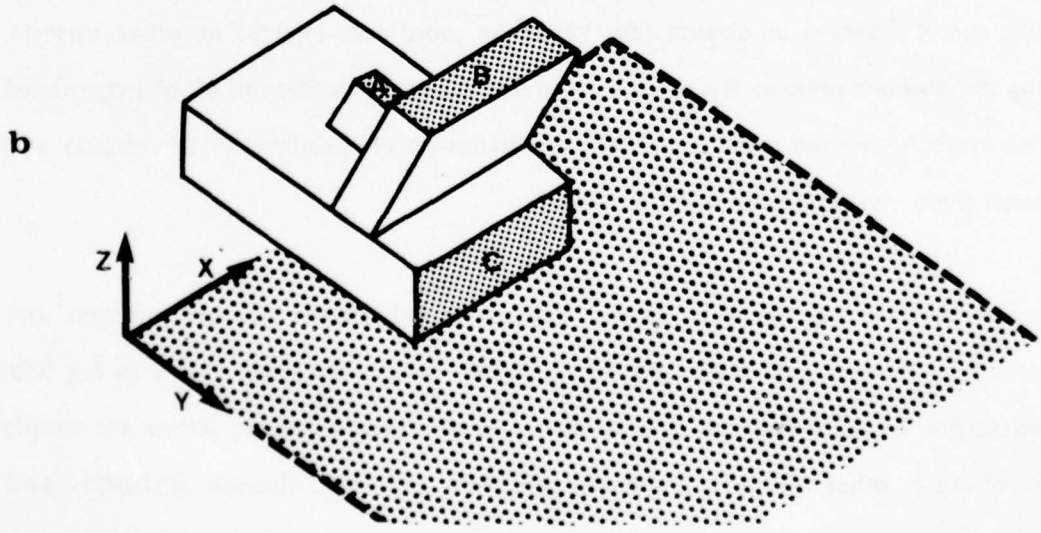


Figure 3.1 - The same block scene (b) is obtained from (a) by rotations. This demonstrates the advantages of relative position descriptors. Both scene have very similar descriptions using relative descriptors but very different descriptions when using absolute coordinates.



orientations are as a descriptive mechanism. Relative positions are much more convenient.

The numerical parameters needed to describe relative positions are still difficult for people to specify reliably. Consider describing that the left face of object A is against the front face of object B. We have to specify the rotation of A that will make the two faces "face" each other and then indicate the displacement of the center of the coordinate systems planted on A and B. It would be much more convenient to be able to specify this directly by means of a symbolic constraint e.g. `(AGAINST (A LEFT) (B FRONT))`.

The rest of this chapter reviews three mechanisms described in the literature for converting symbolic position and orientation constraints into absolute or relative position constraints and presents the scheme used in LAMA.

III.1 Previous Methods

I am aware of three systems implemented to allow the specification of scenes by symbolic constraints such as "Face X is AGAINST Face Y" or "Object A SUPPORTS Object B". The first [Boberg] uses a heuristic generate-and-test strategy to find a configuration that satisfies a given set of constraints. Since that project was carried out in the context of Computer Aided Design it allows more degrees of freedom in the constraints than is necessary in mechanical assembly. For example, Boberg allows the size and shape of objects to be changed. Another system [Ambler & Popplestone] is a part of a mechanical assembly project at Edinburgh [Ambler et al]. Two types of symbolic constraints were considered, *face* AGAINST *face* and *cylinder* FITS-IN *hole*. A constraint between object features is represented as a set of equations involving the rotations and the displacements of coordinate systems based at the features. A more recent system [Taylor], developed in the context of the AL project [Finkel et al], extends the approach in [Ambler & Popplestone]. Taylor's system also represents accuracy information and uses the

constraints between the degrees of freedom to predict maximum variations in the location values.

Boberg's system allows progressive design of a scene compatible with a list of constraints. The constraints he dealt with were very weak, such as LEFT-OF, SUPPORT, etc. These constraints allow many degrees of freedom in the positions and orientations of objects. Boberg was investigating ways to do this type of design by progressive satisfaction of local constraints. His program is rather bulky and inneficient since it must usually attempt several transformations before finding one that generates a consistent scene.

The system reported in [Ambler & Popplestone] takes a completely global approach. Instead of looking for a series of transformation which will reduce the inconsistencies created by a new constraint, as Boberg does, they set up the global equations satisfied by the target state and solve them. This latter method seems more appropriate to the assembly enviroment where the constraints are stronger. The only difficulty is that the equations generated by the constraints are rather awkward. The Edinburgh group has done a very neat job of setting up the equations and of implementing a system to solve them. While they do not describe their implementation in detail, I suspect it is complex, since at the heart of the solution process is an "Algebra System" implemented in POP2.

Taylor's approach is similar in style to that of [Ambler & Popplestone]. He also represents constraints as parameterized mathematical expressions. His mathematical representation is different because it must support the linear-programming type computations he uses to establish bounds on the variations of the parameters.

III.2 A Simple Method

This section presents a simple method to compute the legal ranges of values of the position and orientation parameters of an object, subject to geometrical constraints such as AGAINST and FITS-IN. I have avoided the sophisticated methods introduced by Ambler & Popplestone and extended by Taylor in favor of a simpler and less general method. The domain of objects treated in [Ambler & Popplestone] and in this report is limited to rectangular solids (cuboids) and cylinders. For this important subclass of objects and the type of constraints under consideration, a simple scheme is sufficient.

The polyhedral representation of primitive objects in LAMA is used to model the space taken up by objects. The constraints below deal with the cylinder primitive as a real cylinder rather than its polyhedral approximation. There is no conflict between these different representations.

III.2.1 Terminology

Geometric constraints apply between *features* of objects. In our domain of cuboids and cylinders there are only two types of features: (1) flat *faces* such as the sides of cuboids and the top and bottom of cylinders; and (2) the curved *sides* of cylinders.

Geometric constraints are symmetric in the sense that both objects involved in a constraint relation are constrained. Still, it will prove convenient to use terminology that is asymmetric. Consider the following constraint relation:

R1: The left face of object A is AGAINST the front face of object B

We will refer to object A as the *constrained* or as the *related object*. Object B will be referred to as the *base object*.

The following is a temporary definition of the constraint AGAINST:

> **Two faces are AGAINST each other if they are coplanar with their normals in opposition. The side of a cylinder is AGAINST a face if the axis of the cylinder is perpendicular to the face's normal and the center of the cylinder is a radius' length away from the face.**

This definition is not completely satisfactory, but we will use it for now. We now consider two ways of using the definition above to turn AGAINST constraints into relative or absolute position constraints.

III.2.2 The Equation Approach

One can view constraint R1 as an equation. This equation specifies the transformation necessary to take the position and orientation of the left face of A into that of the (back of the) front face of B. This is a matrix equation with parameters θ, y and z. These correspond to the rotation of A around the normal to its left face and to the translation of A's coordinate system on the front face of B (Fig. 3.2).

We can now introduce another AGAINST constraint:

R2: The bottom face of object A is AGAINST the front face of object C

This constraint would generate a similar equation to the one obtained from R1. Each face has a constant relationship (AT array) to the coordinate system centered in the body it belongs to. From this set of equations a set of constraints on the position and orientation parameters of the bodies can be derived (see [Ambler & Popplestone]).

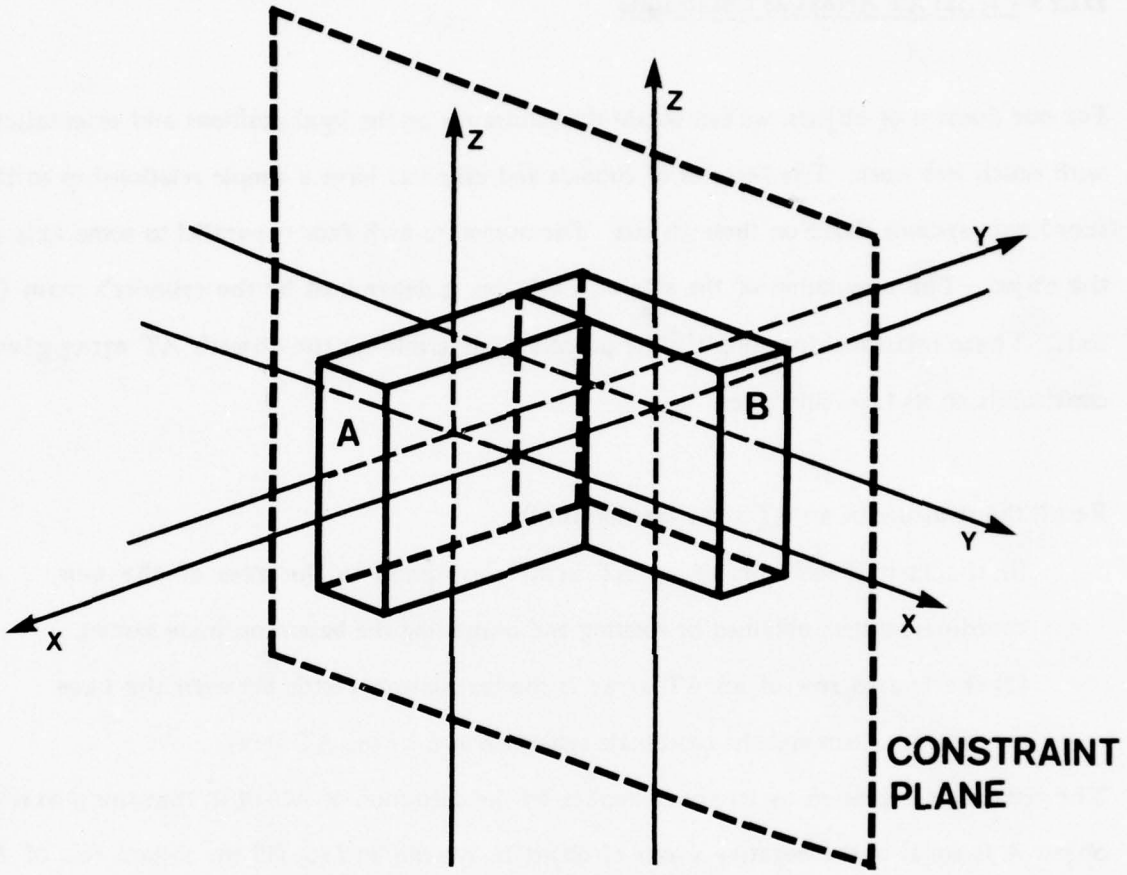


Figure 3.2 - The left face of A AGAINST the front face of B.

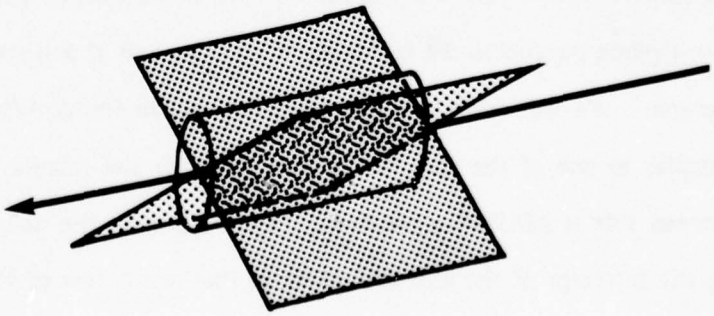


Figure 3.3 - Nonperpendicular constraint planes. This type of situation is not handled by the current implementation.

III.2.3 Partial AT Arrays as Constraints

For our domain of objects, we can obtain the constraints on the legal positions and orientations with much less work. The features of cuboids and cylinders have a simple relationship to the coordinate systems based on these objects. The normal to each face is parallel to some axis of the object. The orientation of the side of a cylinder is determined by the cylinder's main (z) axis. These relationships simplify the process of determining the object's AT array given constraints on its faces and sides.

Recall the structure of an AT array (Section II.2.1):

(1) the first three rows of an AT array correspond to the axes of the new coordinate system obtained by rotating and translating the base coordinate system.

(2) the fourth row of an AT array is the translation vector between the base coordinate system and the coordinate system defined by the AT array.

The constraint expressed by R1 above implies, by the definition of AGAINST, that the y axis of object A is equal to the negative x axis of object B. We can, in fact, fill the second row of A's AT array with the negative of B's first row.

Relation R1 also constrains the relative positions of the two objects. The center of A is restricted to lie on a plane parallel to the two faces in question and at a distance which makes the two faces coplanar. We will call this the *constraint plane* for A. When the constraint plane is perpendicular to one of the axes of the base object of the relation (which is always except when a curved side is AGAINST a flat face), we can represent the position constraint by simply indicating the intercept of the axis and plane in the fourth row of the related object's AT array. If B is a cuboid then the x component of the displacement is half the sum of the corresponding lengths of objects A and B (see Fig. 3.2). It is half the sum of the lengths because the origins of the coordinate systems are located in the center of the cuboids.

Determining A's position and orientation completely would require specifying either the first or third row of A's AT array as well as the y and z components of the fourth row. These three pieces of information represent A's three remaining degrees of freedom, one rotation and two displacements on its constraint plane.

Let us consider the effect of adding relation R2 to the specification of A's position and orientation. As long as the constraints involve faces of cuboids or cylinders there is no interaction between separate constraints on an object. This is because the faces giving rise to the constraints are either parallel or perpendicular to each other so that the constraint planes of an object must either be coplanar or perpendicular. The effect of a constraint perpendicular to some axis of an object is to specify the row of the AT array and the entry in the displacement vector corresponding to that axis. Coplanar constraints generate the same entries; perpendicular ones affect different rows. Thus a new constraint never changes the entries of an AT array determined from other constraints.

A slightly more difficult situation arises when the side of a cylinder is constrained to be AGAINST a face. This constraint leaves two degrees of freedom on the orientation of the cylinder as well as two degrees of freedom on the position. The perpendicularity constraint between the axis of the cylinder and the normal to the face is not sufficient to determine any of the entries in the cylinder's AT array since there is not an unique vector perpendicular to another vector in three-space. To make matters worse, the constraint planes derived from sides of cylinders need not be perpendicular to each other, as is the case for constraints involving faces (see Fig. 3.3). This means that determining the position of a cylinder subject to more than one AGAINST constraints requires the intersection of two nonperpendicular planes. The current implementation only handles perpendicular constraint planes for cylinders.

III.2.4 Some Problems and Some Extensions

The definition of AGAINST we have been using so far is defective in that it allows the configurations shown in Fig. 3.4. There are two reasons for these anomalies. We have treated faces as planes, which they are not. A face is a bounded surface. Two faces are AGAINST each other only if they overlap in area. We have also trivialized the restriction that objects do not occupy the same space. This restriction is only represented in the position constraint between related objects. We have neglected the interactions with other objects in the workspace.

We have been assuming that the representation of the constraints on the generalized position of an object was to be a partially specified AT array. The entries in a partially filled AT array represent the value of the position and orientation parameters whose values are known. We also need to represent the *ranges* of legal values for those parameters which have not been uniquely determined. Ranges of values can provide the mechanism for incorporating the type of bounded restrictions on positions suggested above. These ranges also define the volume in which the presence of other objects would further constrain the position or orientation of the object.

We want the simplest representation of the ranges in the position and orientation of objects. The obvious choice is a list of the min and max values for the angles and displacements of the constrained object's coordinate system relative to some base coordinate system. The choice of the base coordinate system is important to the simplicity of the representation. If the table's coordinate system is chosen, then there is no reason to suppose that the description of the legal values of the position and orientation parameters will fit into simple min-max ranges. In the next section we therefore take advantage of the fact that the degrees of freedom of objects tend to be organized around constraint planes and the lines formed by intersecting them.

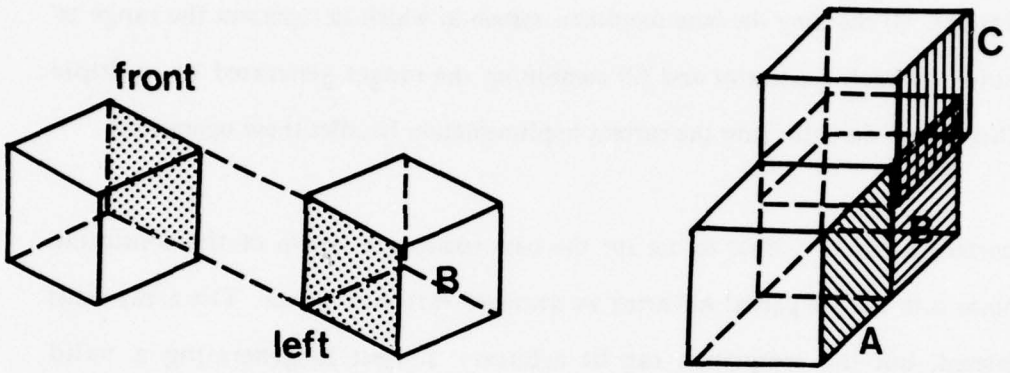


Figure 3.4 - Bugs in the definition of AGAINST. Situation in (a) indicates that faces must OVERLAP as well as be coplanar; (b) shows that the presence of other objects must be taken into account as well.

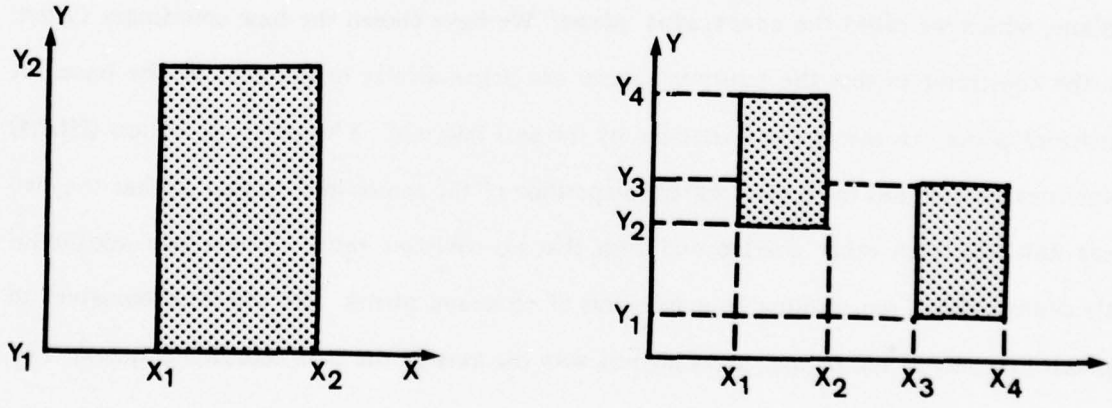


Figure 3.5 - Sets of rectangular ranges of values can be used to represent the ranges of position values consistent with constraints.

III.2.5 An Implementation

An implementation must transform a set of simultaneous constraints on the position and orientation of an object into ranges of values for its position parameters. This involves two important operations: (1) choosing the base coordinate system in which to represent the range of values determined by each constraint and (2) combining the ranges generated by multiple constraints. This section describes how the current implementation handles these operations.

The most important decision is what to use for the base coordinate system of the constraint. One simple choice is to use the partial AT array we discussed earlier as a base. The array must first be completed, but the completion can be arbitrary, subject to generating a valid homogeneous transformation. We can then represent the ranges relative to that coordinate system.

The position of the center of a primitive object subject to an AGAINST constraint is restricted to a plane, which we called the *constraint plane*. We have chosen the base coordinate system for the constraint so that the constraint planes are perpendicular to the axes of the base. A constraint plane can then be characterized by the axis intercept. The previous section (III.2.4) demonstrated the need to further restrict the position of the constrained objects so that the two faces AGAINST each other overlap and such that no collisions result. These two additional restrictions call for representing bounded areas of constraint planes. We will limit ourselves to representing sets of rectangular areas aligned with the axes of the base coordinate system (Fig. 3.5).

The next problem is combining multiple constraints on an object. In general, the range of legal position values of an object subject to simultaneous constraints can be obtained by intersecting the legal ranges of values derived from the individual constraints. In the case of AGAINST

constraints this involves the intersection of the set of rectangular ranges on the constraint planes. Our restriction that constraint planes must be perpendicular to each other (cf III.2.3) makes the intersection operation very simple.

So far, we have considered only ranges of positions limited to constraint planes. We will also need to represent ranges not limited to a plane; for example, restricting an object to be LEFT-OF another. The legal positions of the constrained object form a rectangular volume. This is a straightforward generalization of the method for constraint planes.

The AGAINST constraint also restricts the range of allowable orientations of constrained objects relative to each other. When two flat faces are so constrained, the objects may rotate around the normal to the constraint plane. In the case of a curved side AGAINST a flat face the cylinder is also free to rotate around its central axis. This latter degree of freedom is only important when the cylinder has another object rigidly attached to its side.

In analogy to the ranges of positions, we can represent ranges of angles between an axis of the constrained object and the normal to the constraint plane, which is an axis of the base coordinate system. We can similarly represent the rotation relative to the cylinder's axis. These ranges can be easily transformed into ranges relative to different (orthogonal) base coordinate systems so that they can be intersected with the orientation ranges produced by other constraints.

III.3 Deriving Constraints from Volume Interactions

Each constraint defines a legal set of orientations and positions for the objects involved. This range, in turn, defines a volume of space, the *interaction volume*, obtained by the union of the object's volume at each position in the range. We must determine for each constrained object the intersection of their interaction volumes with those of other objects. The intersection

of these volumes indicates potential collisions between objects. Figure 3.4b shows an example where the range of legal positions of one object is partly determined by the position of another. This section presents a mechanism for finding the locations where objects can be placed such that no collisions arise. This problem is a form of the FINDSPACE problem [Winograd].

III.3.1 Computing the Interaction Volume

The interaction volume of an object is defined by the shape of the part and its ranges of position and orientation. Since objects are represented as unions of cuboids and cylinders, we will limit our discussion to these objects. The interaction volume of a complex object is the union of the interaction volumes of its component parts (ignoring holes).

The interaction volume for a primitive object is derived by first computing the volume taken up the object at a specified position allowing its full range of orientations. This volume is called the *rotation volume* of the object. The space taken up by the rotation volume over the range of positions is the *interaction volume*.

The rotation volume is obtained by first computing the volume taken up by the basic object as it rotates over its range of rotations about x. The resulting volume is then rotated about y and the result of that rotation is further rotated about z. This same procedure is followed for the translational degrees of freedom but starting with the rotation volume.

The method for computing approximations to the *path volumes* needed for this procedure are presented in Chapter 4.

The current implementation then approximates the interaction volume by a cuboid. This cuboid is placed in the model of the environment and any intersections with other volumes are

noted. The intersection volumes themselves are also approximated by rectangular volumes. This simplifies further processing. The intersection operation is described in Appendix 2.

III.3.2 Deriving the Constraints

The procedure described in this section serves to discover locations where a cuboidal approximation to the *rotation volume* (cf III.3.1) of an object can be placed without causing collisions.

The problem dicussed in this section is a close relative of the FINDSPACE problem [Winograd], [Fahlman], [Sussman], and [Pfister]. It is not the general problem because only cuboidal approximations to the obstacles are used and no consideration is given to orientations of the target object. For example, the algorithm described below would not find the position shown in Fig. 3.6. A proposal to extend the current method to handle this case is made in Appendix 2. I know of no efficient solution to the general FINDSPACE problem. Pfister's proposed solution, although probably the most elegant in two dimensions, has similar drawbacks to my algorithm and the extension to three dimensions of his method is awkward.

We will discuss the problem in two dimensions first. This is an important special case. Figure 3.7 shows a typical situation after the volume intersection has taken place. The shaded areas indicate isometric projections of the intersection volumes onto the xy plane. The problem is to compute a range of positions where the two-dimensional projection of the cuboidal approximation to the rotation volume (the *target area*) will fit.

Split the area into strips defined by the vertical sides of the projected intersection volumes (Fig. 3.8). Now, starting at the left, collect adjacent strips into groups whose horizontal dimension is greater than the horizontal dimension of the target area. These strips are then merged such

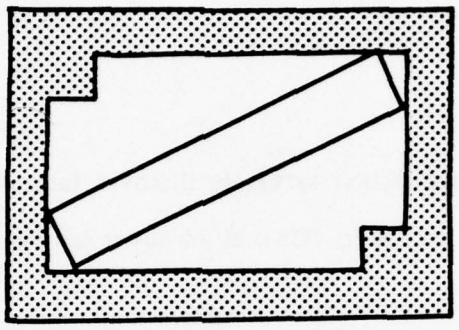


Figure 3.6 - Difficult case for the FINDSPACE computation. The shaded areas are obstacles, the unshaded rectangle is the object to be positioned.

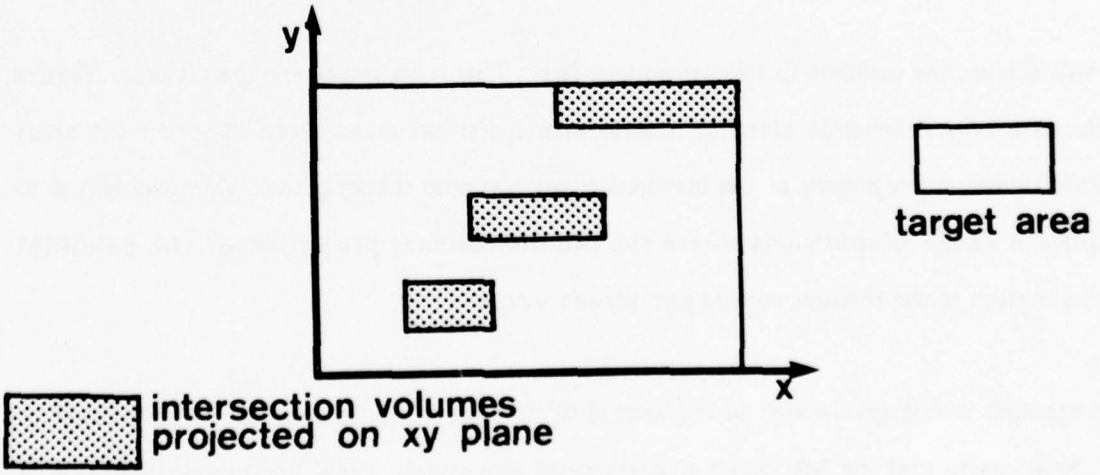


Figure 3.7 - Typical situation after the intersection of an interaction volume has taken place. The shaded areas are the projections of obstacles impinging on the interaction volume. The problem is to find a location where the target area (the rectangular approximation of the rotation volume of the object) can be placed without touching the obstacles.

that adjacent volumes that are unoccupied are coalesced (Fig. 3.9). Any resulting areas whose dimensions are greater than that of the target area is included in the solution. The next step is to remove from consideration the leftmost strip in the group and to add enough strips from the right to bring the horizontal distance occupied by the strips, up to the target area's horizontal length. The merging is done again, and the process repeats. The only exception is that if a strip, whose width is greater than or equal to the horizontal dimension of the target area, is ever encountered, the process starts anew from the next strip after merging the current group with the large strip. The reason for the last step is that such a large strip will complete any areas pending in the group of strips so that only new strips need be considered.

The merging of the strips is not difficult. The vertical strips are divided into horizontal strips along the boundaries of the instersection volumes (Fig. 3.10). If we represent each occupied horizontal strip by a 1 and the empty ones by 0, then the merging operation is an inclusive OR on all the strips. Any empty horizontal strip whose vertical dimension is greater than that of the target area is placed in the solution.

The ranges produced by this process describe the areas where the object can be placed in any of its legal rotations. Some of the areas will overlap as shown in Fig. 3.11, but this is not a real drawback. The only remaining problem is converting these areas into position ranges for the part. This is done by computing the displacement from the min and max value of each of the coordinates of the rotation volume to the center of the object. These displacements are then substracted from the corresponding coordinates of the areas described above. This is guaranteed to produce non-empty ranges since the dimensions of each area is known to be greater than or equal to those of the target area.

The process can be generalized to three dimensions fairly easily. Imagine projecting all the intersection volumes onto the bottom plane of the space. We could then carry out the procedure

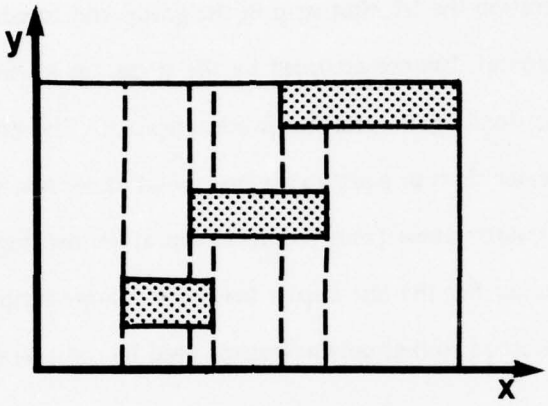


Figure 3.8 - Strips obtained from Fig. 3.7 by extending the vertical sides.

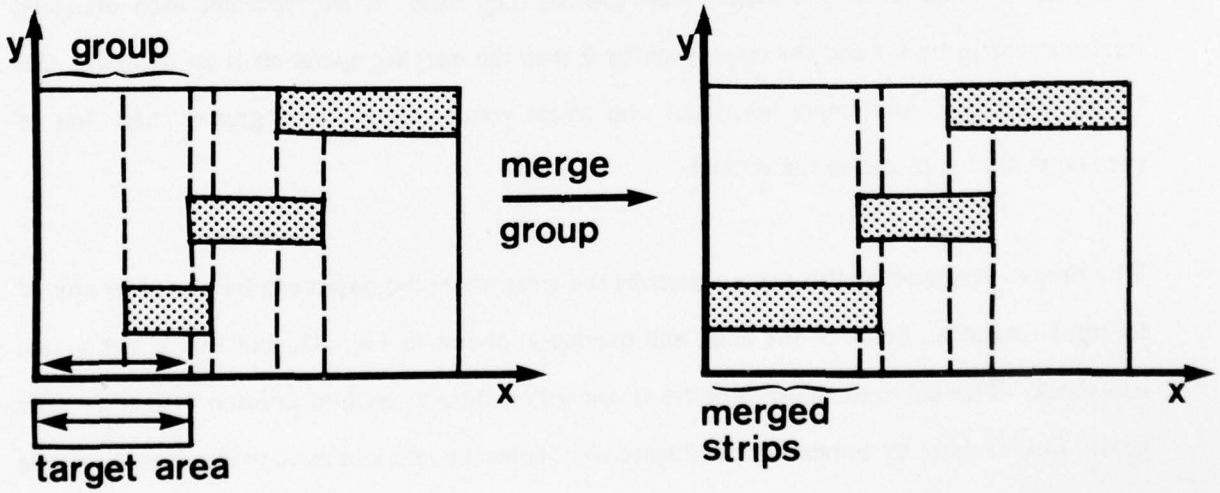


Figure 3.9 - Example of merging a group of horizontal strips.

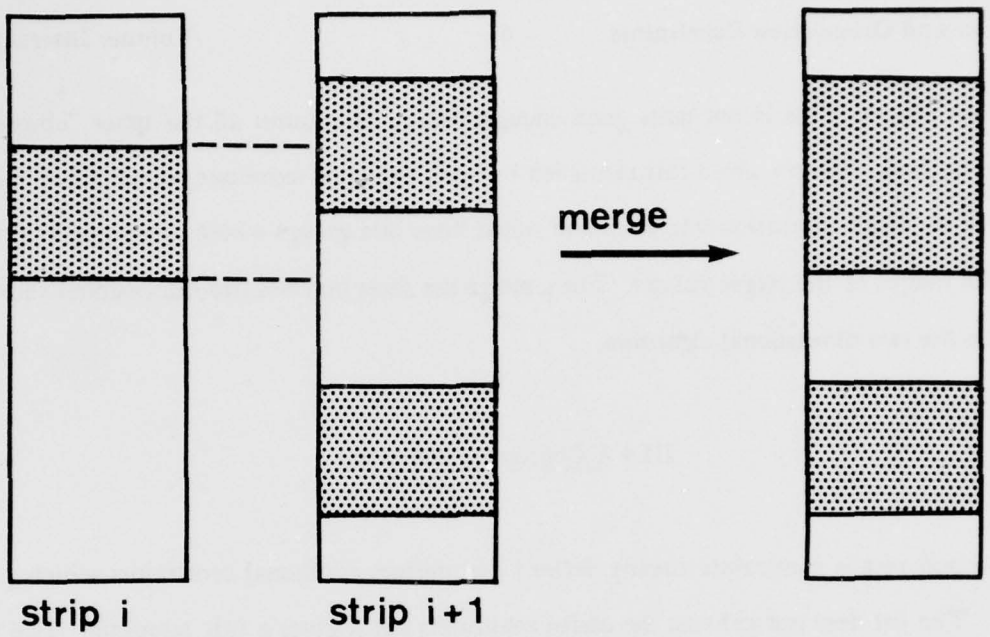


Figure 3.10 - Horizontal strips are merged by an OR operation.

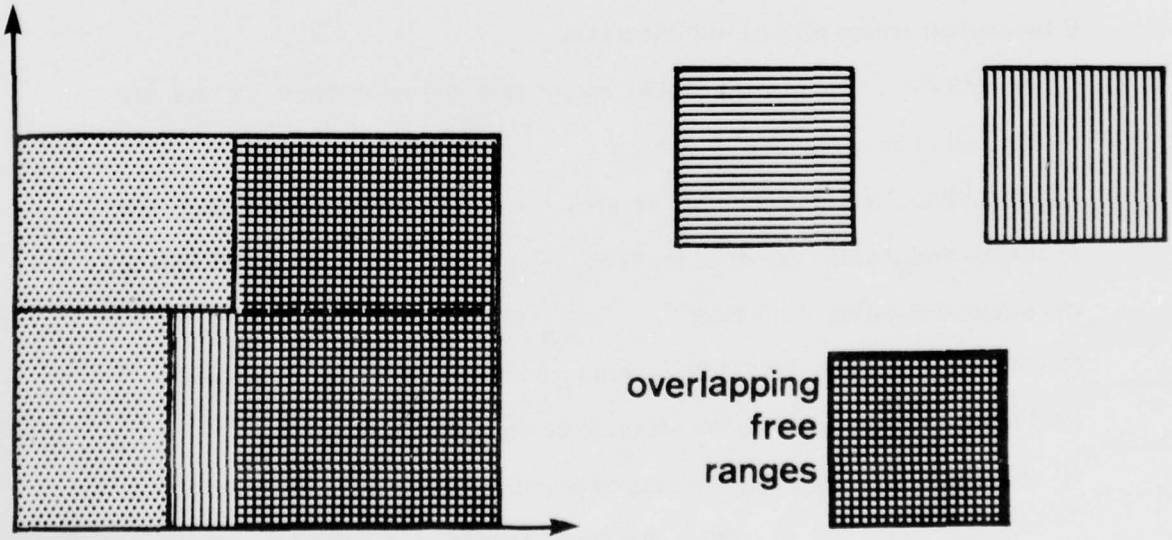


Figure 3.11 - The results of the computation, in general, produces overlapping areas large enough to hold the object.

described above. This is not quite good enough because it requires all the space "above" an area to be free. We can avoid this restriction by using the same technique we described earlier: Segment the third dimension into slices and collect them into groups whose dimension is greater than the height of the target volume. Then, merge the slices into one two dimensional slice and perform the two dimensional algorithm.

III.4 A Constraint Repertoire

This section recalls constraints already defined and defines additional constraints which prove useful. The list does not exhaust the useful constraints but it gives a fair sampling. The term *orientation vector* refers to the normals of faces or the *z* axis of cylinders.

(1) ALIGNED+: It restricts faces to being coplanar with their normals pointing in the same direction. The faces do not necessarily overlap. Aligned cylinders are restricted to have their *z* axes point in the same direction and to have the vector between their centers aligned with the *z* axes.

(2) ALIGNED-: Similar to ALIGNED+ except that the orientation vectors are constrained to be negatives of each other.

(3) ALIGNED&CENTERED: Applies between faces and indicates that they are ALIGNED- and that the centers of the faces can be connected by a vector parallel to the orientation vector of the faces.

(34) FACING+: Features are FACING+ when their orientation vectors are parallel.

(5) FACING-: Restricts orientation vectors to be negative of each other.

(6) OVERLAPS: It restricts plane faces to be placed so that if:

(OVERLAPS FACE1 FACE2)

Then projecting FACE1 along the normal of FACE2 they have a nonnull intersection.

(7) AGAINST: Two faces are AGAINST each other if they are coplanar and their normals are opposed. The faces are required to overlap. A cylinder's side is AGAINST a face when the orientation vectors of the cylinder and the face are perpendicular and the origin of the cylinder's coordinate system is the length of the radius away from the face.

```
(AGAINST FACE1 FACE2)
```

is equivalent to

```
(AND (ALIGNED- FACE1 FACE2) (OVERLAPS FACE1 FACE2))
```

(8) CONTACT: This applies between an object and another object or a surface (face or side) and says that they touch. This is a weaker form of AGAINST.

(9) FITS-IN: An object FITS-IN a hole if the hole and the object are either ALIGNED+ or ALIGNED-; their axes OVERLAP and the object's maximum radius is less than that of the hole.

(10) WEAK-LEFT-OF, WEAK-RIGHT-OF, etc.: Indicates a relationship between two objects, e.g. in WEAK-LEFT-OF it constrains the leftmost part of an object to be left of the leftmost part of another object.

(11) LEFT-OF, RIGHT-OF, etc.: Indicates e.g. that one object's rightmost part is left of another object's leftmost part. These constraints can also be applied to pairs of positions such as objects' centers.

(12) BETWEEN: Applies between three positions, e.g. centers of objects, and specifies that the projection of the first onto the line formed by the other two lies between the endpoints of the line segment.

Each of these constraints is subject to the additional restriction that no two (positive volume) objects should share any volume of space as described earlier.

In addition to the geometrical constraints described above, certain simple motion constraints are defined. These constraints indicate that motion is possible in a specified direction.

(1) FREE-TO-MOVE: This constraint specifies an axis and an optional range of values. The meaning is that the object can move over the range of value without causing a collision or violating any other constraints.

(2) FREE-TO-ROTATE: Like FREE-TO-MOVE but specifies rotations.

Of course, the other constraints also implicitely define ranges of allowable motions.

III.5 Combining and Modifying Constraints

Constraints can be combined and modified by the Boolean operators AND, OR and NOT. The operators map into intersection, union and complement of the ranges specified by the objects. The modifiers PARTLY and ALMOST serve to define ranges of positions close to the ones specified by the exact constraints above.

The modifier PARTLY only applies to the constraint FITS-IN and specifies a range of values of the distance of the constrained object inside the hole.

ALMOST applies to the constraints ALIGNED+, ALIGNED-, FACING+, FACING-, OVERLAPS, AGAINST and CONTACT. It specifies a range of angles for the ALIGNED and FACING constraints and a range of distances for the others.

III.6 The Uses of Constraints

The repertoire of geometric constraints available to the system is used in a number of contexts:

(1) Indicating the relative positions of the primitive objects (cuboids and cylinders) composing a part. This was discussed in Chapter 2.

(2) Describing positions in the assembly description and the assembly plan. The constraints serve to indicate the initial and desired positions and orientations of the objects involved in the assembly steps. Chapters 4 and 6 will expand on this.

(3) One of the components in the description of the assembly operators available to the assembly planner is a list of the prerequisites and initial positions of objects in the operation. These prerequisites are a description of the desired position, orientation and motion constraints on the objects involved in the operation. These will be discussed in Chapter 6.

(4) Describing the results of assembly operations. The basic assembly operations can be catalogued and described by means of the constraints they impose on the objects they operate on. This description can be used to simulate the operations in a model of the workspace. Chapter 5 explains this application in detail.

IV. The PICK AND PLACE Problem

This chapter deals with the following problem, which we will refer to as the *pick and place* problem:

Given:

(1) An object description.

(2) An initial position and orientation.

(3) A target position and orientation.

(4) A spatial model of the environment.

Find:

(1) A way to grasp the object at both the initial and target positions.

(2) A collision free path to the initial position and from there to the final position.

The program to be described below actually solves a generalization of the grasping problem as stated above. It computes not just a way of grasping the object but a large class of legal grasp positions.

IV.1 An Overview of the Approach

At one level of description the problems of grasping and path calculation have a common formulation. They are both the problem of placing an object, subject to a set of spatial constraints, so that no collisions result. In grasping, the object to be placed is the manipulator's hand. In path calculation, it is the volume swept out by the hand and object between two positions. In fact, our generalization of the grasping problem involves the volume swept out by the hand over a range of legal grasp positions.

Stating the problem as one of avoiding collisions should make it clear that our basic tool is a method of discovering those collisions. Since objects are described as a composition of rectangular and cylindrical volumes, finding collisions involves intersections of these basic volumes. In fact, we will only deal with the intersections of polyhedra and use a polyhedral approximation to the cylinder primitive (Fig. 4.1). We use a program (Appendix 2) that, given a complex object (composed of polyhedra) at a given position in the environment, returns an approximation to the intersection of the object with any solid objects impinging on its volume. The description of an intersection consists of a list of the min-max *xyz* values (each a box aligned with the axes) of the intersection volume (Fig. 4.2).

The intersection operation treats each of the primitive objects separately. The intersection volume shown in Fig. 4.2 corresponds to the intersection of two cuboids. Complex objects can generate many min-max *xyz* pairs reflecting the pairwise intersection of its component objects with the environment. Note that each primitive object, being convex, can only generate one intersection volume when intersected with another primitive object.

IV.1.1 Grasping

There are two generically different classes of grasping positions on an object. In the first the fingers are in contact with surfaces of more than one of the basic objects which comprise the object description. The second type restricts the fingers to touch only one of the component objects. Throughout this report we will ignore grasp positions where the fingers touch more than one of the primitive objects.

When grasping an object, the system examines the legal ways of grasping each of the basic volumes (cuboids and cylinders) composing the object and computes which of these positions are reachable and cause no collisions. This approach amounts to first using a simple theory, i.e.

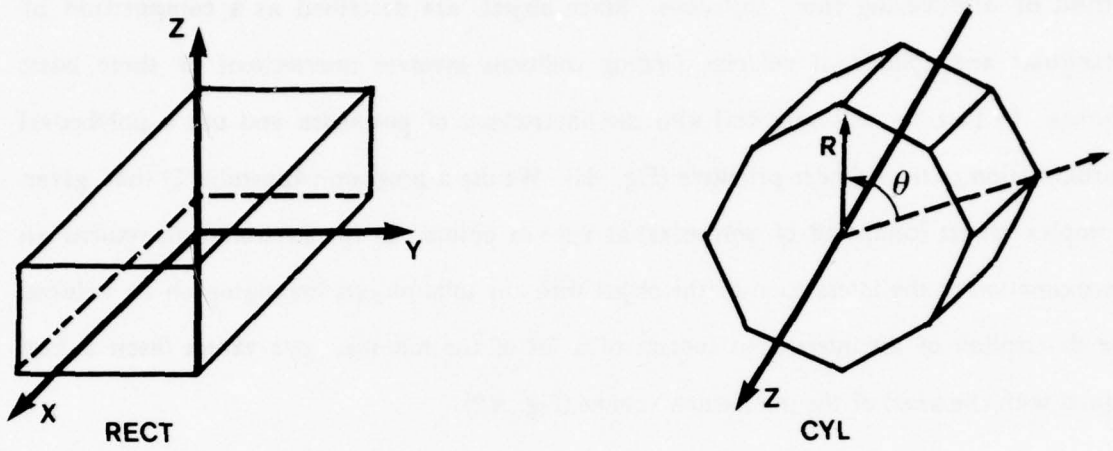


Figure 4.1 - Primitive objects in the system models. Same as Fig 1.4.

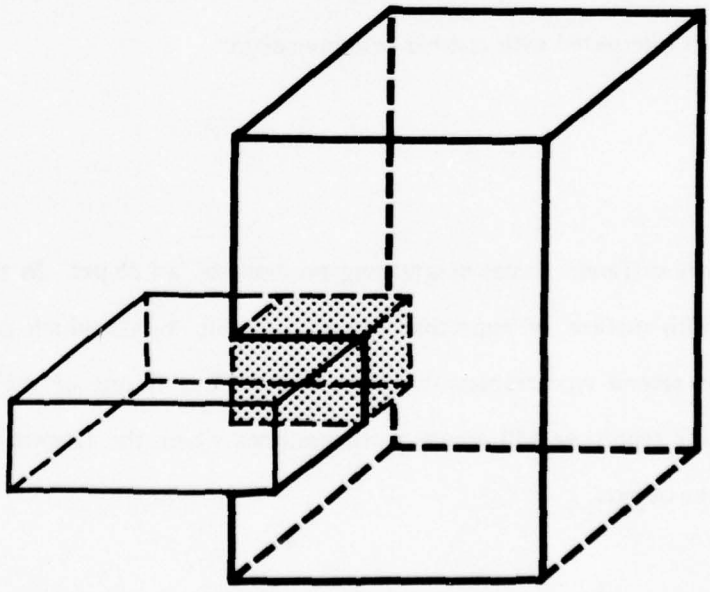


Figure 4.2 - Intersection example. The result of the intersection of volumes is a set of rectangular solids aligned with the global coordinate axes.

grasp one of the component objects as if the others were not there; and then debugging the theory, i.e. remove from consideration any grasp positions giving rise to collisions.

Each object type has a small number of different ways in which it can be grasped. One way differs from another mainly in that different surfaces are involved in the grasping operation. For each surface (or surface pair) there is a range (set) of positions where the hand can be placed. These ranges can be parameterized by the displacements and/or rotations, relative to the surface, needed for the hand to reach a position on the surface. We will call each of these parameterized ranges a *grasp set*. Later, we will see how to describe the legal grasp positions on the primitive object types using only four types of grasp sets.

The term "set" is used because the operations of union, intersection and set-difference are defined for grasp sets of the same type. This raises the problem of how to describe the legal grasp positions such that set operations are meaningful. One solution is to represent them as ranges of positions, relative to coordinate systems located on the object to be grasped. Each type of grasp set has a local coordinate system that standardizes the meaning of the coordinates in the ranges. For a given object and grasp set these ranges can be unioned, intersected and differenced.

The generalized grasping problem can be solved by finding the legal position ranges on each of the object's grasp sets, first in the initial and then in the target environment, and intersecting these sets. The result of the intersection is the range of legal grasp positions on the object in both environments.

This approach to grasping, computing the ranges of legal grasp positions, contrasts to a *generate-and-test* approach, which would entail suggesting a grasp point and testing whether it is reachable without collisions. I believe the "grasp set method" to be preferrable for

two reasons:

(1) The grasp set method is likely to be more efficient. The major computational cost of the grasping computation operation is proportional to the number of intersections that are performed. A technique that relies on trial and error is likely to perform more intersection operations.

(2) By computing ranges of positions the grasp set method provides the information needed for the choice of a good grasp point, rather than merely an adequate one.

IV.1.2 Collision Free Trajectories

The computation of collision free trajectories can be divided into (1) finding the collisions and (2) avoiding them. Collision detection is done by (a) calculating the volume swept out by the hand (and whatever it is holding) and (b) intersecting this volume with the environment. Collision avoidance consists of generating a trajectory that avoids the objects identified during the detection phase. The only collision avoidance strategy we will consider is the trivial one of always trying to rise above the obstacle.

IV.1.3 Pick and Place

We can now solve the *pick and place* problem as stated in the introduction to this chapter as follows:

(1) Compute the ranges of legal grasp positions of the object in the initial position.

(2) Do the same for the destination.

(3) Intersect the ranges found in (1) and (2).

(4) Find the "best" legal grasp position in the intersection.

(5) Compute a collision free trajectory to this grasp position at the initial position of the object.

(6) Calculate a path from the initial to the final position of the object.

Finding a grasp position in the initial environment and the trajectory to it are not completely independent operations. If the position is not reachable, a new grasp position must be chosen.

IV.2. Uncertainty in Pick and Place

The Pick and Place phase serves to tie together the assembly steps. Each step constrains the original and final positions of the parts involved in it. The Pick and Place operation is responsible for placing the parts in the intial position required for each assembly step, such as insertion. We have, so far, assumed that both the position required for the step and the original position of the part were known. That is not always the case. When first grasping a part from a table or pallet, its position is subject to some uncertainty. Also, a step might not specify the initial position of a part completely, thus allowing succeeding operations to provide the necessary constraints. This avoids making unnecessary decisions but also makes computing a grasp point and a collision-free path more difficult.

This chapter will assume that the object to be grasped has negigible amounts of uncertainty in its orientation and position. Chapter 5 will briefly discuss grasping strategies that tolerate some uncertainty in the position and/or orientation of the objects. The extension of these methods to parts with significant uncertainty is an interesting problem for further research.

Since the initial position of the parts in an operation is not known until the operation is instantiated by the Feedback Planner, the path computation is postponed until the operation has

been expanded. Similarly, there is no single destination at which ranges of legal grasp positions are to be computed (step 2 in section IV.1.3); rather there is whole sequence of motions which have to be considerd. Grasping during these motions is treated very similarly to grasping at one specified point; the only difference is that the volume the hand occupies over the grasp set is projected along the path of the motion. The grasp ranges for each of the motions is computed as it is being considered by the Feedback Planner. After all the motions have been expanded the ranges are intersected together with the grasp ranges for the object at its initial position. Only then is the final grasp position chosen.

The presence of error and uncertainty has another effect on the Pick and Place computation. The volumes used for detecting contacts are not simply the volume of the parts involved but the *interaction* volume (cf. v.2.1). This volume is defined as the union of the volume of the part at all the positions and orientations it might have. Thus, an intersection between two interaction volumes means that a contact might occur. This becomes important during the collision avoidance computation, since some obstacles might not always be avoidable.

IV.3 Legal Grasp Positions (GSETS)

The sets of legal grasp positions for an object must be specified compactly. For example, all the grasp positions reachable by sliding the fingers along the parallel surfaces of a cuboid are potentially valid. We cannot hope to represent them all separately, not even at some coarse resolution. Clearly, we want a method of describing the whole range with a few parameters. It is equally clear that this description should not be in terms of absolute hand coordinates. This would imply that the description of the legal grasp positions for an object would change when the object moved. The solution is to describe the legal hand positions by specifying the position of the finger-tips relative to the object's coordinate system. In fact, for a cuboid, we can describe it relative to a coordinate system specific to each pair of parallel faces. This reduces

the problem to one of specifying sets of rectangular areas aligned with the axes (Fig. 4.3).

The real problem is deciding what portion of the range of legal positions is lost due to an obstacle in the environment. I will refer to this process as *pruning a grasp set*. We will see that the description of grasp sets simplifies the pruning process.

We will consider four classes of legal grasp positions for the domain of rectangular and cylindrical solids. Each of these will be called a *gset*. There will be a systematic ambiguity to the use of this term. It will be used to describe the abstract classes of grasp positions, the range of positions that a class implies on a particular object and the actual reachable subset of legal positions for the object. The particular use should be clear from context.

IV.3.1 The Linear GSET (LGSET)

The simplest grasp set is the linear gset (LGSET). It describes the set of legal grasp positions obtained by translations over a rectangular surface (Fig. 4.3) while maintaining a constant orientation of the hand. The legal grasp positions are expressed relative to a coordinate system local to the gset, shown in Fig. 4.3. The LGSET restricts the fingers' z axis to be parallel to the LGSET's y axis. This restriction makes the LGSET less than fully general since it disallows rotation with respect to the LGSET's z axis. We will see some of the implications of this later.

An LGSET corresponds to one side of a rectangular surface pair of a cuboid. Consider any two parallel surfaces on a cuboid. Each side of those surfaces can accomodate an LGSET. Thus, there are potentially twelve LGSETs on a cuboid. The coordinate system for the LGSET is centered at the object's center and is specified as a transformation on the object's coordinate system.

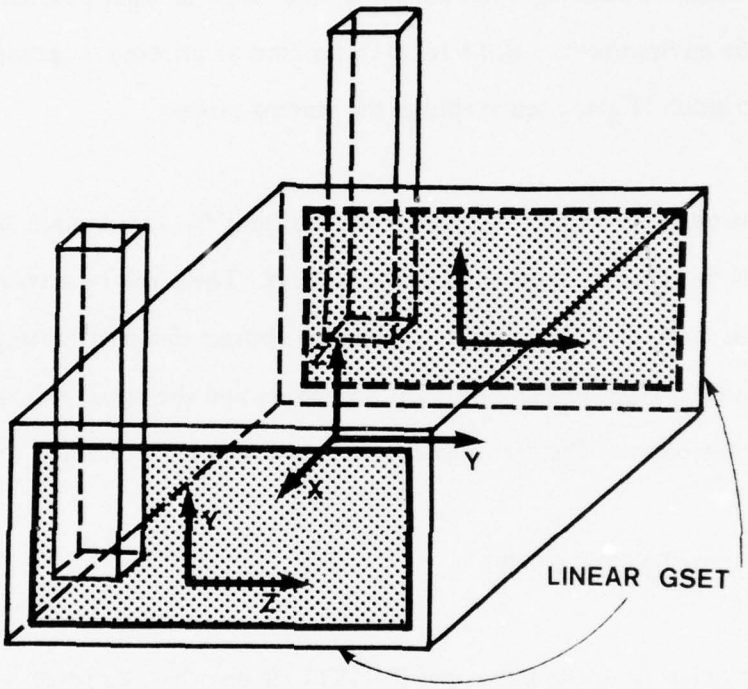


Figure 4.3 - LGSET. An LGSET describes the range of legal grasp positions along one side of a surface pair of a cuboid. Each LGSET is really two surfaces (shown shaded) defined by the legal locations of the finger tip. Each LGSET has a local coordinate system (shown).

IV.3.2 The Polar GSETs (PGSET)

The other major class of gsets, the PGSETs, are each characterized by one rotational and one translational degree of freedom. There are three types of PGSETs. Fig. 4.4 shows them in relation to a cylinder.

PGSET[1] corresponds to grasping a cylinder which is standing on end. This allows rotation around the z axis of the cylinder and displacement along the length of the axis (up to the length of the fingers).

PGSET[2] describes the range of positions corresponding to grasping a cylinder lengthwise (with a finger on each end). PGSET[2] allows radial displacement and rotation about the cylinders central axis.

PGSET[3] restricts the fingers to be along the side of the cylinder with the fingers perpendicular to the cylinder's axis. It is generated by rotation and translation along the object's z axis.

IV.2.3 Putting the Gsets Together

The four types of gsets can be combined to describe most of the legal grasp positions for cuboids and cylinders. A cuboid is composed of three parallel surface pairs. On each surface pair we can place an LGSET along each of the sides and a PGSET[2] at each corner (Fig 4.5). Notice that the each PGSET[2] has its θ range restricted to $\pi/2$. PGSET[1] and PGSET[3] are undefined for cuboids. This arrangement, although imcomplete, seem to capture a large subset of the legal grasp positions, while allowing the pruning computation to be relatively easy.

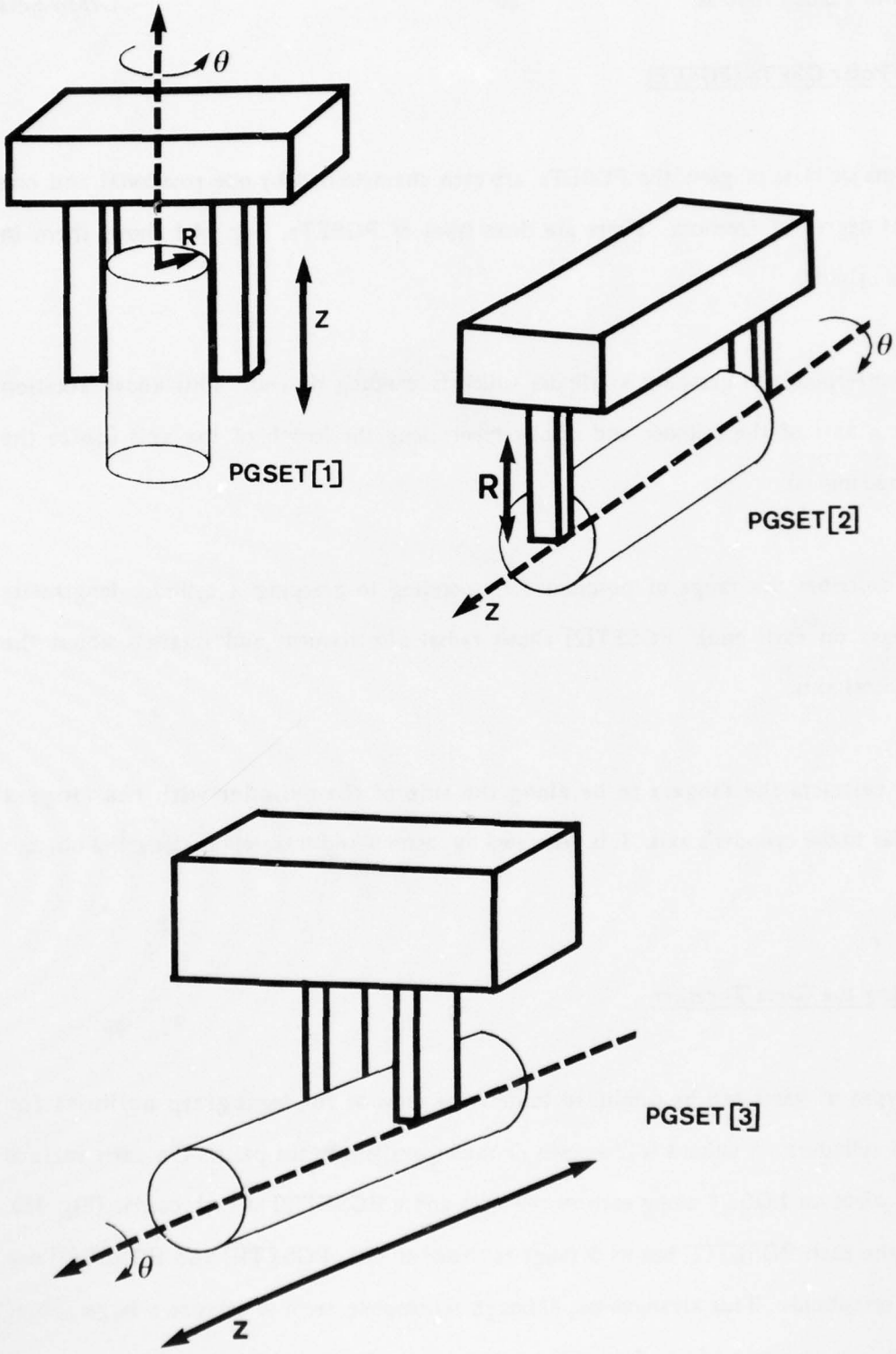


Figure 4.4 - PGSETs. There are three PGSETs indicating the three legal ways of grasping a cylinder.

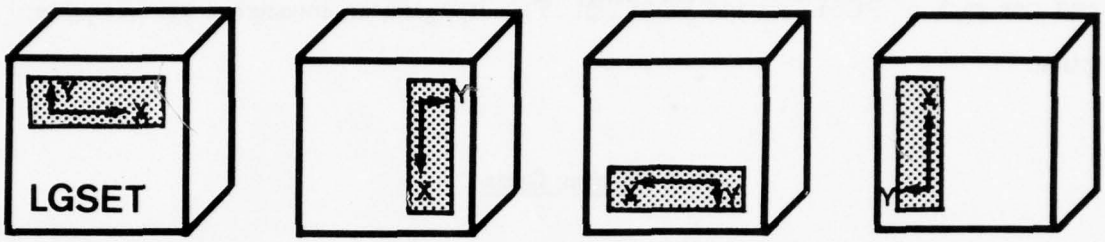
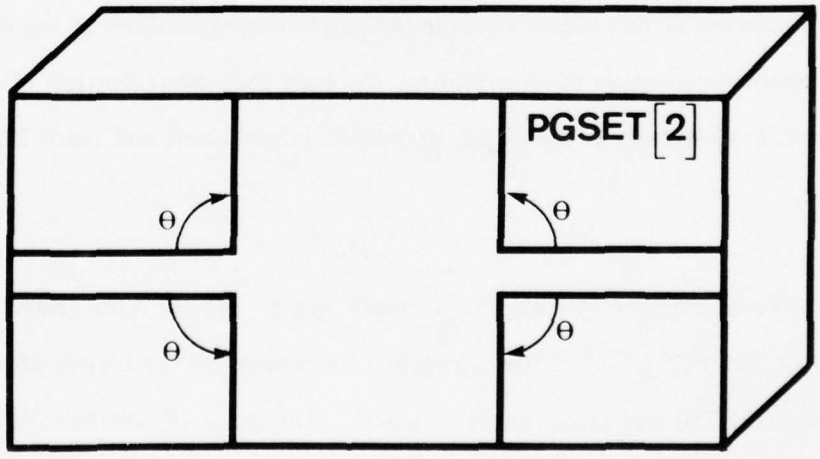


Figure 4.5 - Putting the GSETs together. This indicates how four LGSETs and four PGSET[2]s (with θ restricted from 0 to $\pi/2$) are combined to cover most of the legal grasping positions on one surface pair of a cuboid.



The legal grasp positions for a cylindrical object are described by two PGSET[1]'s (one at each end) and one each of PGSET[2] and PGSET[3]. This is, again, an incomplete yet adequate description.

IV.4. Pruning Gsets

A gset defines a domain of legal grasp positions, actually a surface of legal positions for the finger-tips. For the whole gset to be accessible, the manipulator hand must be free to be at each of the positions defined by the gset. This set of positions defines a volume (the projection of the hand's volume over the grasp surface) which must be free of other objects. Because we are considering each component object individually as a potential "handle", many otherwise legal grasp positions imply interference between the hand and other parts of the object to be grasped. Any part of the environment might block access to a whole or part of an object's surface. Whole gsets are inaccessible due to limitations in the positioning capabilities of the manipulator. The pruning operation consists of determining, for each gset, what portion of the gset is reachable given the other parts of the object, the current environment and reach limitations on the manipulator.

Figure 4.6a shows a hypothetical object in empty space. Figure 4.6b shows a graphic representation of the PGSET[1] of the cylindrical component of the object after pruning. Notice that the PGSET[1] has three ranges of values. One range (RANGE1) allows grasping the full length of the cylinder. Another range shows (RANGE2) that the fingers can only go as far down as the cuboid connected to the cylinder. Another range of values (RANGE3) is ruled out altogether by the interference of the fingers with the second cuboid.

The pruning operation requires a model of the manipulator which, together with the gset, defines the interaction volume of the gset. This is the volume to be examined for obstacles.

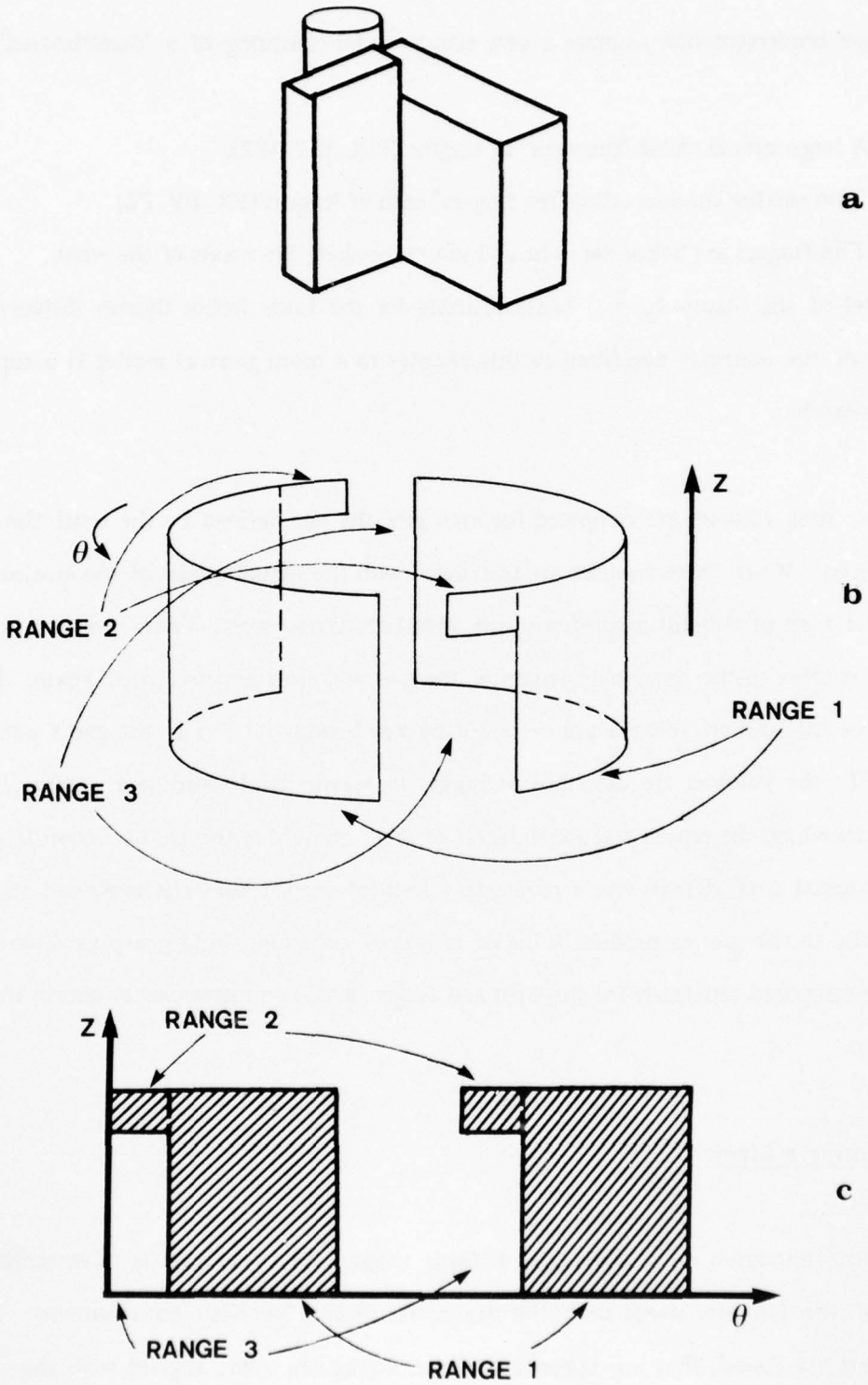


Figure 4.6 - PGSET[1] example. (a) is the object to be grasped, we consider grasping the cylinder. (b) shows the surface of legal positions for the fingers. (c) is a planar representation of (b).

The current implementation assumes a very simple model consisting of a "disembodied" hand (Fig. 4.7):

(1) A large cuboid called "the wrist" of lengths (WX, WY, WZ).

(2) Two smaller cuboids called "the fingers" each of lengths (FX, FY, FZ).

(3) The fingers are below the wrist and can move along the x axis of the wrist.

This model of the manipulator is fairly accurate for the Little Robot System [Silver]. The extension of the methods described in this chapter to a more general model is a topic for further research.

In practice, three volumes are computed for each gset, the one defined by the wrist, the others by the fingers. When these volumes are intersected with the spatial model of the environment, the result is a set of cuboids aligned with the global coordinate axes. These volumes are then expressed relative to the coordinate system of the gset and then approximated again. For an LGSET, the intersection volumes are described by xyz bounds relative to the gset's axes. For the PGSETs the volumes are described as ranges in a cylindrical coordinate system (r, θ, z) (Fig. 4.8) in which the pgset's rotational degree of freedom defines the angular coordinate and the translational d.o.f. defines the z parameter. Each of these volumes is processed in a way characteristic to the gset to produce a list of ranges of remaining legal grasp positions. The ranges are computed separately for the wrist and fingers and then intersected to obtain the final gset ranges.

IV.4.1 Pruning a Linear GSET

The pruning operation on a linear gset is fairly simple. The first step is to instantiate the volume of the fingers, swept over the grasp set, in the specified environment. If any intersections are found, they are approximated by rectangular solids aligned with the axes of the gset. We can treat the intersection volumes produced by both fingers together since an

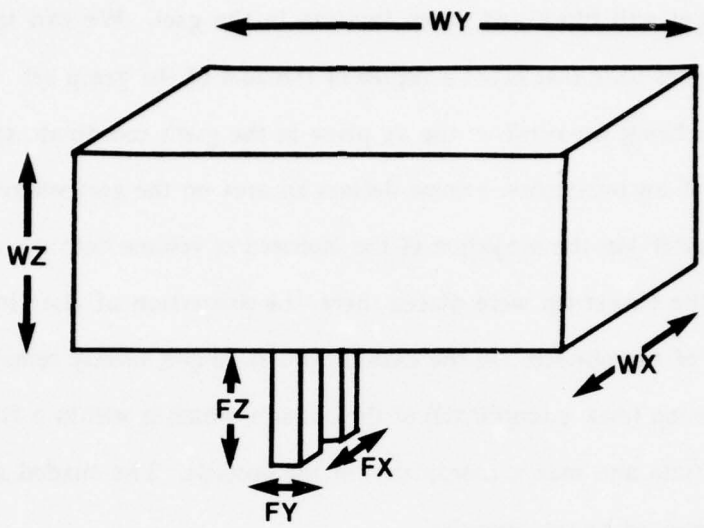


Figure 4.7 - Disembodied hand model with dimensions.

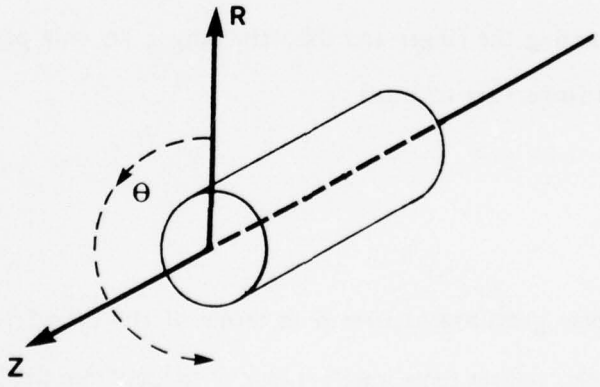


Figure 4.8 - Cylindrical coordinates.

obstacle to either finger will block any given location in the gset. We can ignore the z coordinates of the volumes since that is not a degree of freedom of the grasp set. Thus, let us consider the cuboids as being projected on the xy plane in the gset's coordinate system. Each rectangular projection of an intersection volume defines an area on the gset where the fingers cannot be. The area is not just the projection of the intersection volume onto the xy plane. It is the area where, if the finger-tip were placed there, the projection of the fingers would overlap the projection of the obstacle. In the LGSET this is simple, merely remove from the gset any area below the top (max y coordinate) of the obstacle, which is within a finger's width away from either side (min and max x coordinates) of the obstacle. The shaded areas in Fig. 4.9 illustrate the area removed by two obstacles.

This operation is done once for the intersection volumes obtained from intersecting the volume swept out by both fingers along the gset and once for the wrist volume. This in turn generates two sets of ranges of finger-tip positions. The one set is obtained from examining the legal positions of the fingers and the other from examining legal wrist positions. The resulting ranges are obtained by intersecting the finger and the wrist ranges, i.e. only positions that can be reached by both wrist and fingers are returned.

IV.4.2 Pruning PGSET[1]

The legal ranges of the polar gsets are expressed in terms of the cylindrical coordinates described above. Each obstacle removes some specific range of values from the legal ranges.

The pruning operation for the fingers in PGSET[1] is the "polar" equivalent of the LGSET operation. Think of the domain of legal positions as split into a set of ranges, along the min and max θ values of each of the intersection volumes. For each θ range we remove the z range below the max z value of the intersection volumes in that θ range. This means that any finger

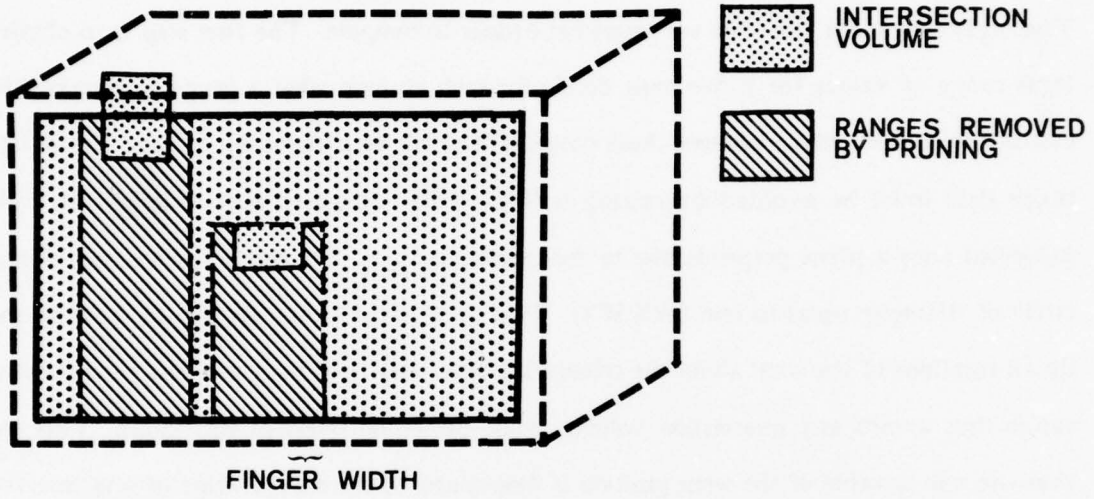


Figure 4.9 - Pruning an LGSET. The dotted areas are volumes obtained from the intersection with the environment of the finger volume convolved with the LGSET range. The striped areas are the ranges of the LGSET removed by these obstacles.

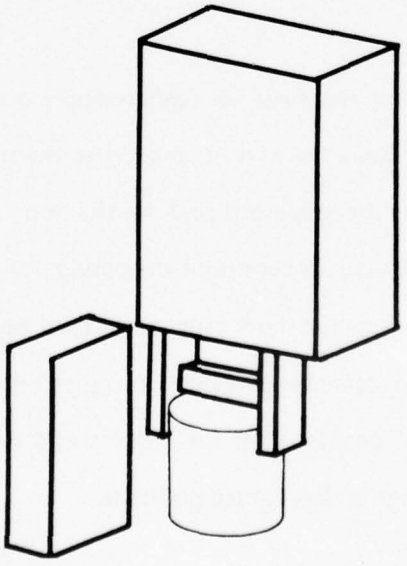


Figure 4.10 - Two ways of avoiding obstacles in PGSET[1]: (1) Going above the piston and rotating the wrist to avoid the block next to it.

position that overlaps the θ range of an obstacle is constrained to have a min z value greater than the max z for that obstacle. All angles are normalized to between 0 and π radians to account for the symmetry in the hand positions.

The legal ranges for the wrist are somewhat trickier to compute. The first step is to obtain the legal range of values for z. We must decide for each obstacle what z range is inaccessible. A distinction is to be made between those obstacles that can be avoided by rotating the wrist and those that must be avoided by raising it (Fig. 4.10). Consider each intersection volume projected onto a plane perpendicular to the cylinder's z axis. On this plane we can identify a circle of diameter equal to min (WX,WY). This circle is the crossection of the volume shared by all rotations of the wrist about the cylinder's z axis. Any legal wrist position must occupy a z range that avoids any intersection volume whose projection overlaps this circle. This means that the min z value of the wrist position is determined by the max z value of any intersection volume in the circle. Any other obstacles, outside the circle and whose z ranges go above the min z value computed above for the wrist, can be avoided either by rotating the wrist or by moving it up. We will now consider how to avoid obstacles by rotating the wrist.

To compute the legal θ ranges for the wrist we cannot simply discard the θ range for each obstacle in the relevant z range. Since the axis of rotation of the wrist goes through its center, we cannot accurately characterize the space occupied by the wrist in terms of the θ range it occupies (Fig. 4.11). Each obstacle actually constrains the position of the outer edges of the wrist (see Fig. 4.12). From this we can compute the θ ranges that must be discarded, as shown in Fig. 4.12. These operations must be performed with values of angles between 0 and π to account for symmetries. The range of legal positions for the fingers and those for the wrist are then intersected to obtain the final range of legal grasp positions.

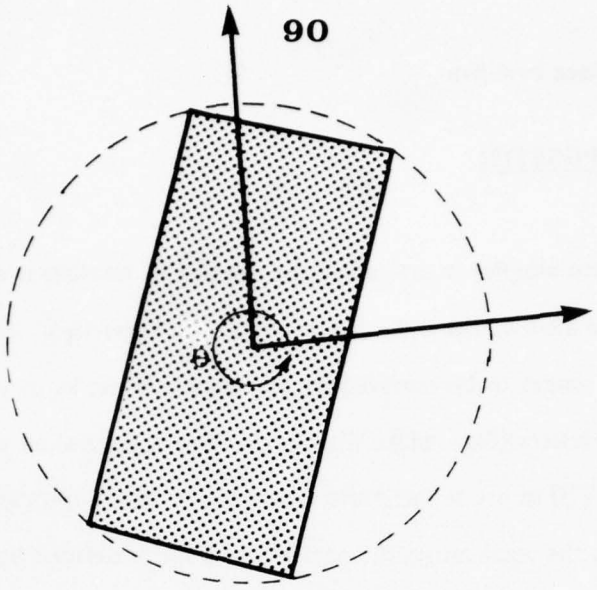


Figure 4.11 - The rectangle in the figure occupies space over a range of 2π in angle but a solid circle is a bad approximation for it.

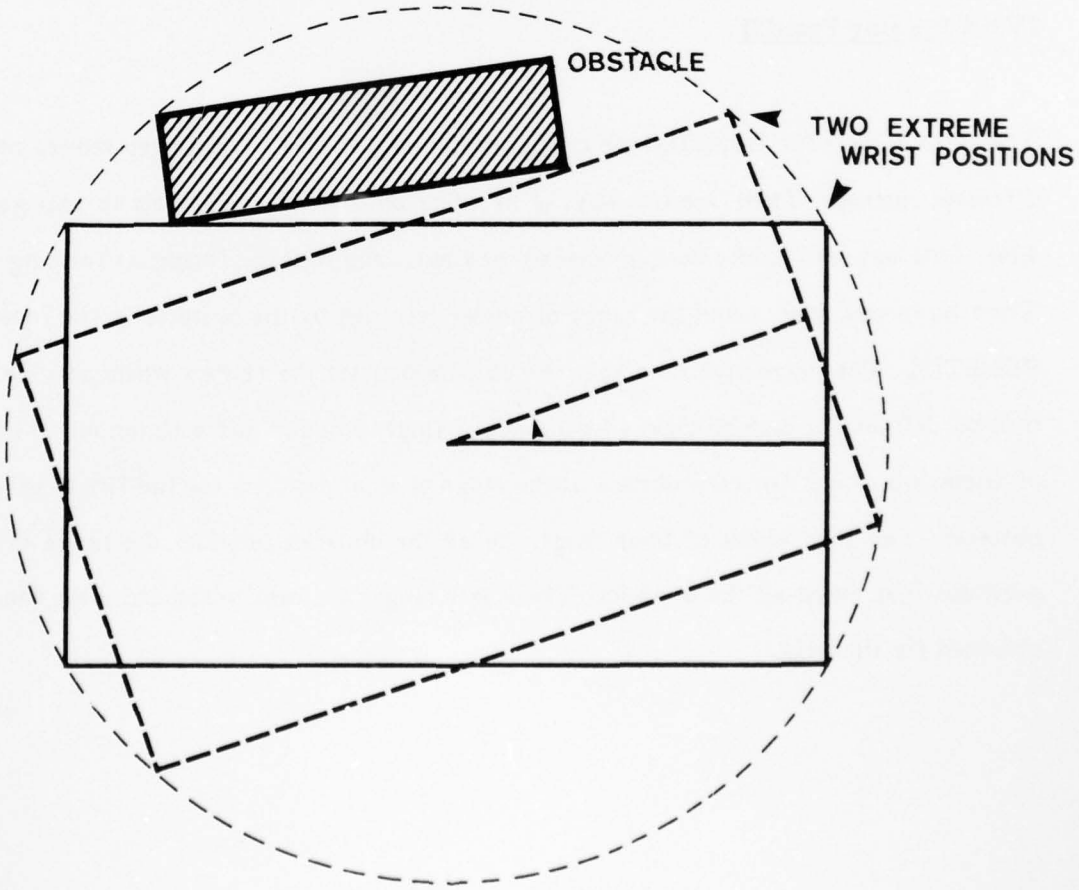


Figure 4.12 - An obstacle in PGSET[1] constrains the position of the outer edges of the wrist. The range of legal orientations lies between the orientations that cause contacts.

IV.4.3 Pruning PGSET[2]

PGSET[2] is much simpler to prune because the axis of rotation is external to the manipulator. Namely, it is the axis of the cylinder between the finger-tips. This condition allows us to characterize the ranges to be removed by an obstacle, in terms of the bounds of the r and θ values of their vertices (Fig. 4.13). We can perform an operation similar to that used for the fingers in PGSET[1] to prune the r and θ ranges for both fingers and wrist in PGSET[2]. We can remove from the legal range, the part of the θ range defined by each obstacle, where the r coordinate is less than the max r coordinate of the obstacle (Fig. 4.13). The ranges for the wrist and for the fingers are then intersected.

IV.4.4 Pruning Pgset[3]

The wrist ranges for PGSET[3] are computed as in PGSET[2]. The finger ranges require a different method. There are two ways of avoiding obstacles to the fingers in this gset (Fig. 4.14). One way is to ignore the gap between the fingers and treat the fingers as forming a solid. Then have this solid avoid the range of angles occupied by the obstacle in the manner of PGSET[2]. The alternative is to keep the obstacle between the fingers when possible. Each method defines, for each obstacle, a legal range of finger positions and orientations. The union of these two ranges for each obstacle is the range of legal positions for the finger given that obstacle. The intersection of these ranges for all the obstacles provides the range of finger positions that avoid all the obstacles. The finger ranges are then intersected with the ranges obtained for the wrist.

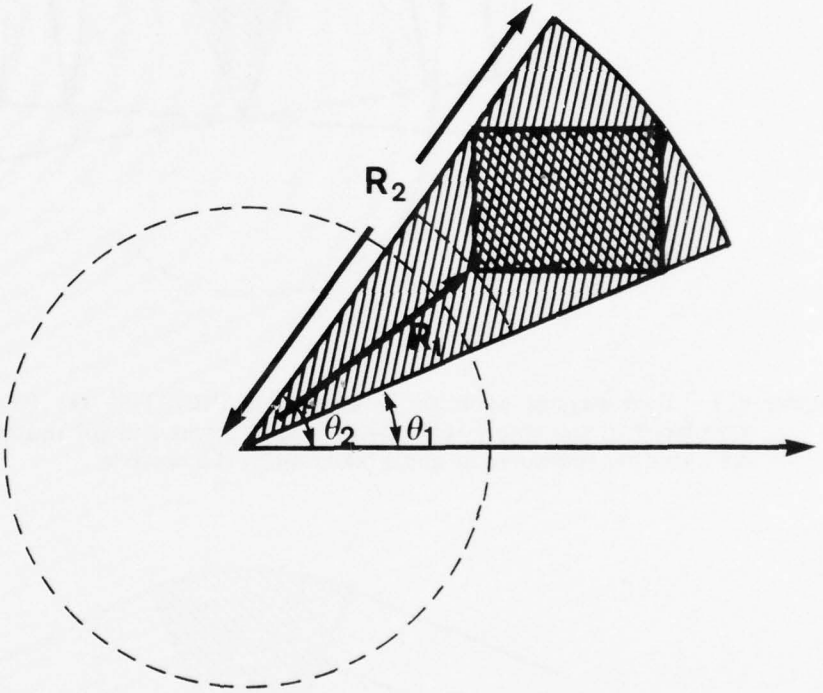


Figure 4.13 - In PGSET[2], obstacles are characterized by the range of r and θ values of their vertices.

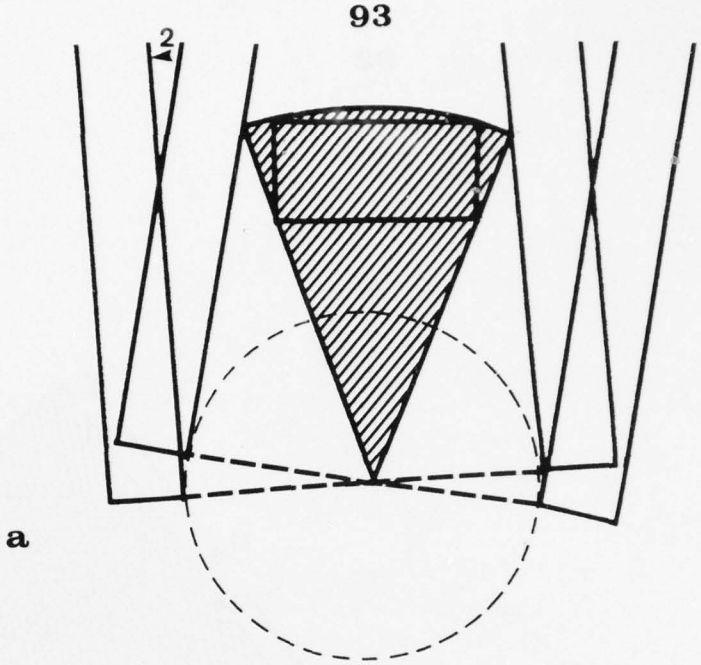
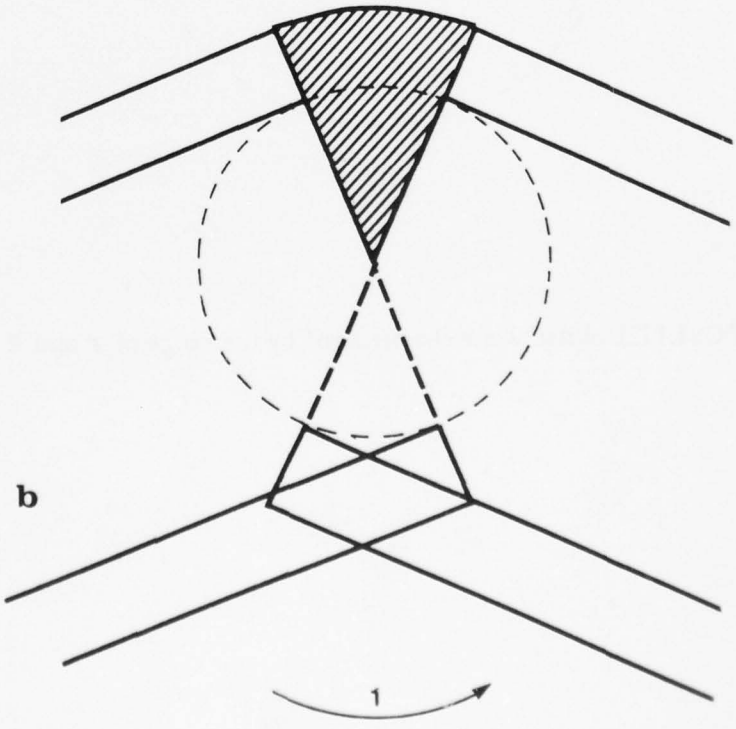
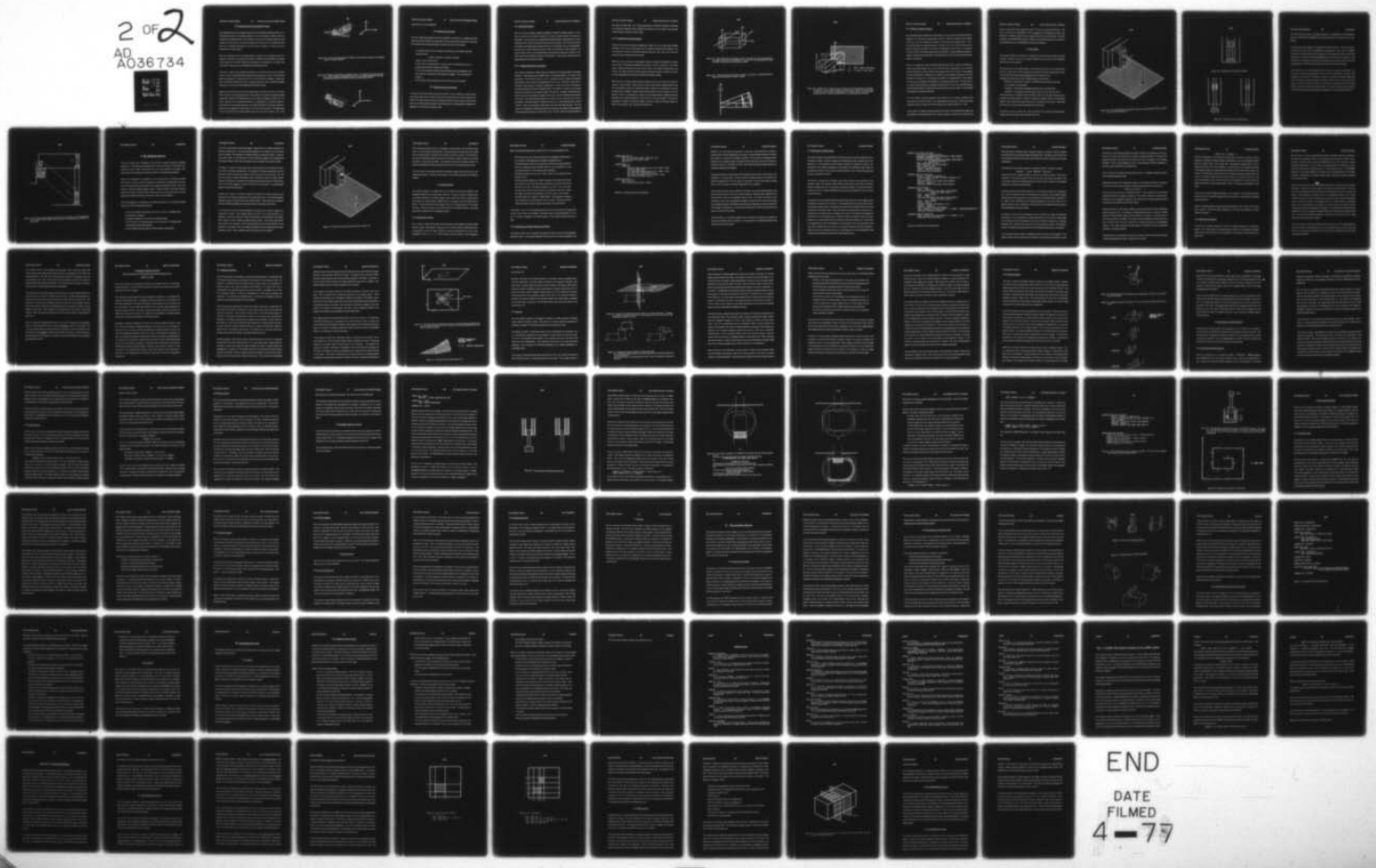


Figure 4.14 - Two ways of avoiding an obstacle in PGSET[3]: (a) shows how the obstacle can be placed in the empty space between the fingers and (b) shows how it can be avoided by avoiding the range of angle taken up by the obstacle.



AD-A036 734 MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTE--ETC F/G 9/2
THE DESIGN OF A MECHANICAL ASSEMBLY SYSTEM.(U)
DEC 76 T LOZANO-PEREZ
UNCLASSIFIED AI-TR-397 N00014-75-C-0643
NL

2 OF 2
AD A036734



END
DATE FILMED
4-77

IV.5 Tailoring the Gsets to the Little Robot System

The manipulator used in the research reported here, the Little Robot System [Silver], is not a fully general position and orientation generator. It has five degrees of freedom which are divided in the following manner: (1) an xy table, (2) a wrist which can displace along and rotate around the z axis and (3) a vise which rotates about the y axis (Fig. 1.2). This means that over most of its workspace (excepting the vise) it has only 4 degrees of freedom, the three translations and rotation about z.

Since the manipulator lacks two rotational degrees of freedom, two of the polar gsets are not necessary to represent the reachable grasp positions. PGSET[2] and PGSET[3], for cylinders, can be replaced by an LGSET whose y axis lies parallel to the manipulator's z axis (Fig. 4.15). I will refer to these grasp sets as LGSET[2] and LGSET[3] respectively.

In order for a GSET to be accessible, the principal axis of the GSET (the y axis in an LGSET and the z axis in a PGSET[1]) must be aligned with the manipulator's z axis. This means that given the orientation of an object, only a few of its GSETs might be accessible. A cuboid has only two LGSETs accessible at any time. An upright cylinder has only its PGSET[1] available while one on its side has only the LGSET[2] and LGSET[3] accessible.

Objects which don't have at least one axis perpendicular or parallel to the hand's z axis (Fig. 4.16) will not have any legal grasp positions. This problem can be ignored for general purpose manipulators because the manipulator hand can be rotated to compensate for the tilt in the object. Because of the rotational limitations in our manipulator, this situation requires a generalization of the pruning operation. The ranges removed by an obstacle should be a function of the orientation of the gset relative to the hand's coordinate system. The dependance can be characterized by the angle between the gset's main axis and the hand's z axis. This

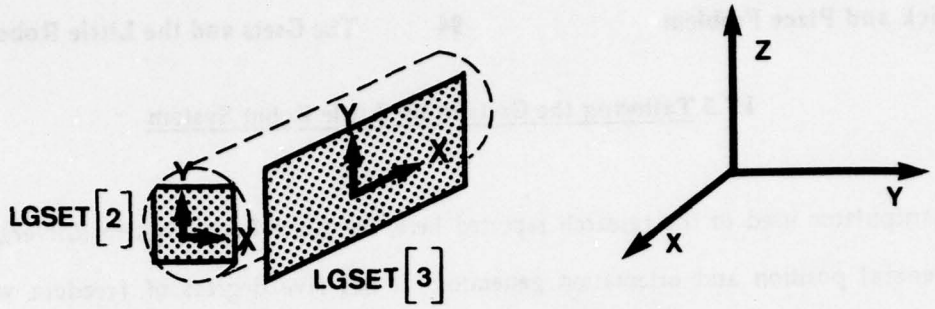
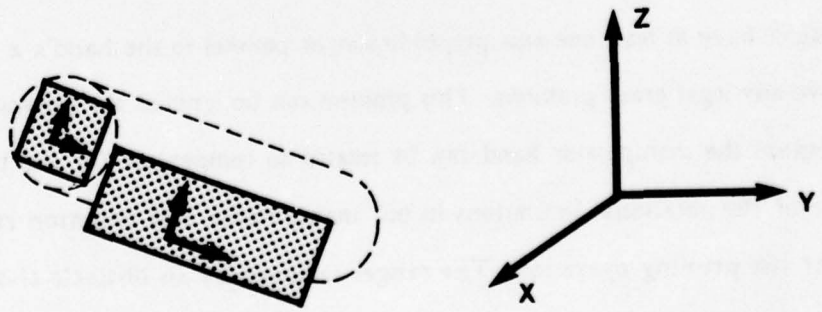


Figure 4.15 - In the Little Robot System the PGSETs become LGSETs because of the limited degrees of freedom available.

Figure 4.16 - Problems with LGSETs on unaligned objects. If an object does not have an axis that is either parallel or perpendicular to one of the manipulator coordinate axes then the legal grasp positions of the object cannot be described using LGSETs.



generalization was not implemented.

IV.6. Choosing a Grasp Position

The set of legal grasp positions can now be computed for the object at its original and final positions and these ranges can be intersected. From this range of grasp locations we must pick one particular place to grasp the objects. Several factors enter into this decision.

(1) The gsets have an a-priori preference ordering based on how good a hold they typically provide:

LGSET > PGSET[2] > PGSET[1] > PGSET[3]

Where > means "is preferred to".

(2) The area of the legal portion of the gset affects the likelihood that errors in positioning will hinder the approach to the object.

(3) The forces generated by contacts during assembly generate non-negligible forces and torques on the grasp point which might cause slippage. This is considered in Chapter 5.

(4) The distance from the grasp point to the center of gravity of the object.

IV.7. Collision Detection and Avoidance

To insure that the path between one point and another is clear of obstacles, we must intersect the volume swept out by the hand and its contents along the path, with the other objects in the model. This section discusses first how to compute the volume swept out along a path and then how to avoid the obstacles so detected. Since all objects are to be represented as collections of polyhedra, we will limit our discussion to these.

IV.7.1 Swept Out Volumes

The first step in the collision avoidance computation is locating the possible collisions. To do this we must know the volume of space that is swept out along the path from origin to destination. I do not know of a computationally efficient solution to the problem in its exact mathematical form. But, of course, we do not need an exact solution. We do not need to know the shape of the volume swept out along a path to an accuracy greater than of the manipulator. We also want to avoid "close calls" and must allow for uncertainties in object positions. Yet another factor, is that we are actually interested in identifying the objects near the path of the hand rather than determining the shape of the intersection. All this seems to justify the use of approximations to the swept out volumes.

IV.7.1.1 Volumes Swept Out by Translations

The volume swept out by an object under pure translation can be approximated very simply. Consider a vector going from the object's center at the original position to its center at the destination. Let this vector be the z axis of a coordinate system. Choose two mutually perpendicular vectors which are also perpendicular to this z axis. These three vectors form a coordinate system centered at the object's original position. Project the vertices of the object in its original position into this new coordinate system. The pattern of vertices on the xy plane defines the crossection of the swept out volume. We will use a rectangular approximation in which the sides of the rectangle are aligned with the axes of the new coordinate system defined by the path. This approximation amounts to using the min-max xy bounds of the vertex coordinates. There only remains to define the min and max z values along the path. Clearly, the min z value of the projected vertices serves as the min value along the path. The max value can be computed by adding the distance from the origin to the destination (the magnitude of the translation) to the max z value of the vertices. This new cuboid is an approximation to

the swept out volume (Fig. 4.17). A better approximation is possible by finding the minimum area enclosing rectangle [Freeman & Shapira] that includes all the xy values of the projected vertices instead of using the min-max ranges.

IV.7.1.2 Volumes Swept Out by Rotations

There are two cases to be considered, depending on whether the axis of rotation goes through the object or not. The first step, in either case, is to represent the object by the ranges in (r, θ, z) covered by its vertices in a cylindrical coordinate system (Fig. 4.18). The z axis of this new coordinate system lies along the axis of rotation.

When the axis of rotation does not go through the object, the figure then described is a section of an annulus. The annulus is described by the range of r, θ and z covered by it. The radial range is obtained directly from the min and max values of the cylindrical coordinates of the vertices. The angle range can be readily computed by adding the angle of rotation to the max or min θ (depending on the direction of rotation) of the object's verteces.

When the axis of rotation passes through the object, the vertices can be split into those for which $\theta > \pi$ and the others. These two sets of vertices are then treated as separate objects, each of which generates a section of a cylinder upon rotation. Because we are using only the vertices to approximate the object's volume, this split does not guarantee that the z ranges of the new objects are correct. In principle, the z range of the new objects should be determined by the intersection of the plane defined by the z axis and the $\theta = \pi$ line with the object. To avoid this computation we adopt the conservative strategy of using the z range of the original object for each of the new objects. Fig. 4.19 shows an example of this.

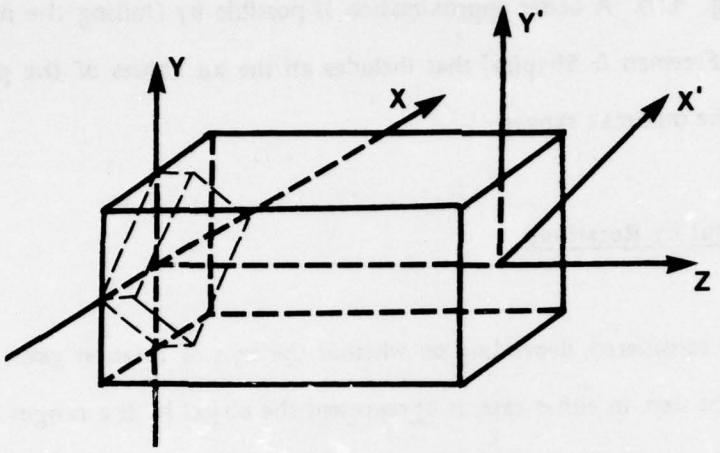
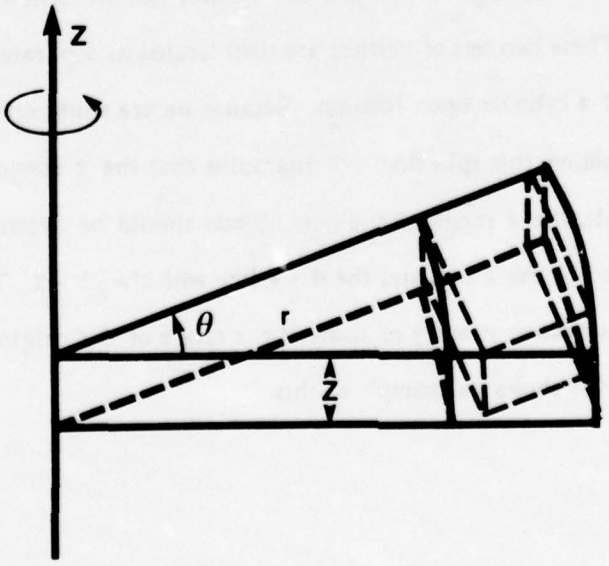


Figure 4.17 - Volume generated by translating a cuboid. The objects are first approximated by an aligned cuboid and the translation is computed on the approximation using coordinate axes aligned with the direction of motion.

Figure 4.18 - Volume generated by rotation of cuboid. The objects are approximated by sections of annuli centered at the axis of rotation.



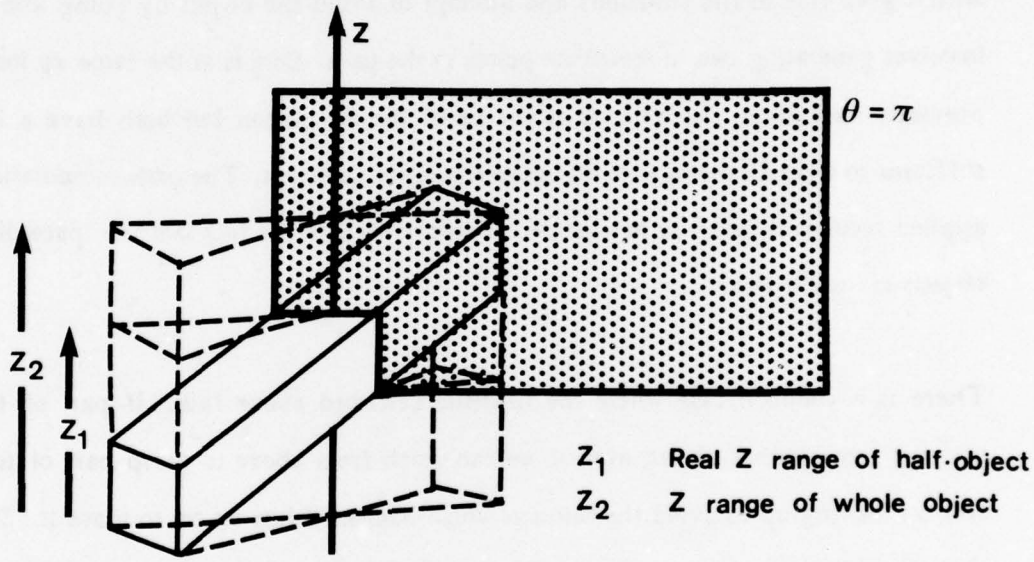


Figure 4.19 - Problem with z range when axis of rotation goes through object to be rotated. The object is first cut in half by a plane with $\theta=\pi$ then the z range of the whole object is used when each half object is approximated by a cylinder section. This avoids the computation of the z values of the intersection of the cutting plane and the object.

IV.7.2 Collision Avoidance Strategies

Once potential contacts (collisions) have been detected, a new path which avoids them must be computed. We will adopt a simple strategy. Identify a cuboid that includes all the objects which give rise to the collisions and attempt to avoid the object by going above it. This involves generating two intermediate points in the path. One is at the same xy location as the previous start point, the other is at the xy of the destination but both have a z coordinate sufficent to clear all the objects that gave rise to the collisions. The path computation process is applied recursively to these new points. This process uses the fact that the space directly above objects is usually clear.

There is a common case where the heuristic described above fails. If part of the object is inserted into another object, even if we can reach from above to grasp part of it; we cannot rely on moving up to avoid the collision which happens when we try to move it. This suggests that the techniques that are adequate for the rough free-flight trajectory computation need to be modified for close-up work. I have ignored this problem in the implementation. The tecniques for solving it are straightforward extensions of the methods presented here. The main idea is to consider alternate motions in the collision avoidance process. We can avoid considering some of the possible directions of motion by examining the constraints on the object. I have avoided the temptation of delving into this topic.

For any collision avoidance algorithm, some collisions are not always avoidable when uncertainties are taken into account. This usually happens near the target. The Pick and Place computation merely records this for further analysis by the Feedback Planner (cf. v.2.2).

At first glance, the collision avoidance problem seems similar to the FINDSPACE problem and the problem of deriving constraints from volumes considered in section III.3. In both cases we

want to avoid contacts with volumes in the space. There are two important differences. The first is that whereas in FINDSPACE we want a position that simultaneously satisfies all the constraints, in collision avoidance we want a path that successively avoids obstacles. The other difference is that FINDSPACE specifies a volume where the desired position is to be found, the path problem does not; it merely specifies the origin and destination.

IV.8 An Example

This section considers the insertion of the piston-rod onto the piston-pin during the piston assembly. The goal is to find a way to grasp the piston-rod and to bring it to the destination defined by the insert operation.

We assume that the rod is held upright on the table and that the pin is in the vise with the piston on it (Fig. 4.20). The first operation is to determine the choice of grasp points for the rod. Each of the primitive objects composing the rod is considered in turn:
Grasping the rod's pin-end at the rod's original position is considered first:

PGSET[1] -- Not accessible. In order for a PGSET[1] to be reached the main axis of the cylinder has to be upright.

LGSET[2] -- This LGSET is completely accessible, as Fig. 4.21 clearly shows.

LGSET[3] -- This position causes the fingers to contact the rod's shaft-end above.

The remainining gset, LGSET[2], will be one choice available to the Feedback Planner when it simulates the insertion operation. During the simulation the Feedback Planner will immediately predict a contact between the fingers and the sides of the piston and with the pin. This means that the pin end cannot provide the grasp point for this operation.

The rod's bar suffers a similar fate. Only two LGSETs are accessible and both generate collisions between the fingers and the shaft-end (Fig. 4.22).

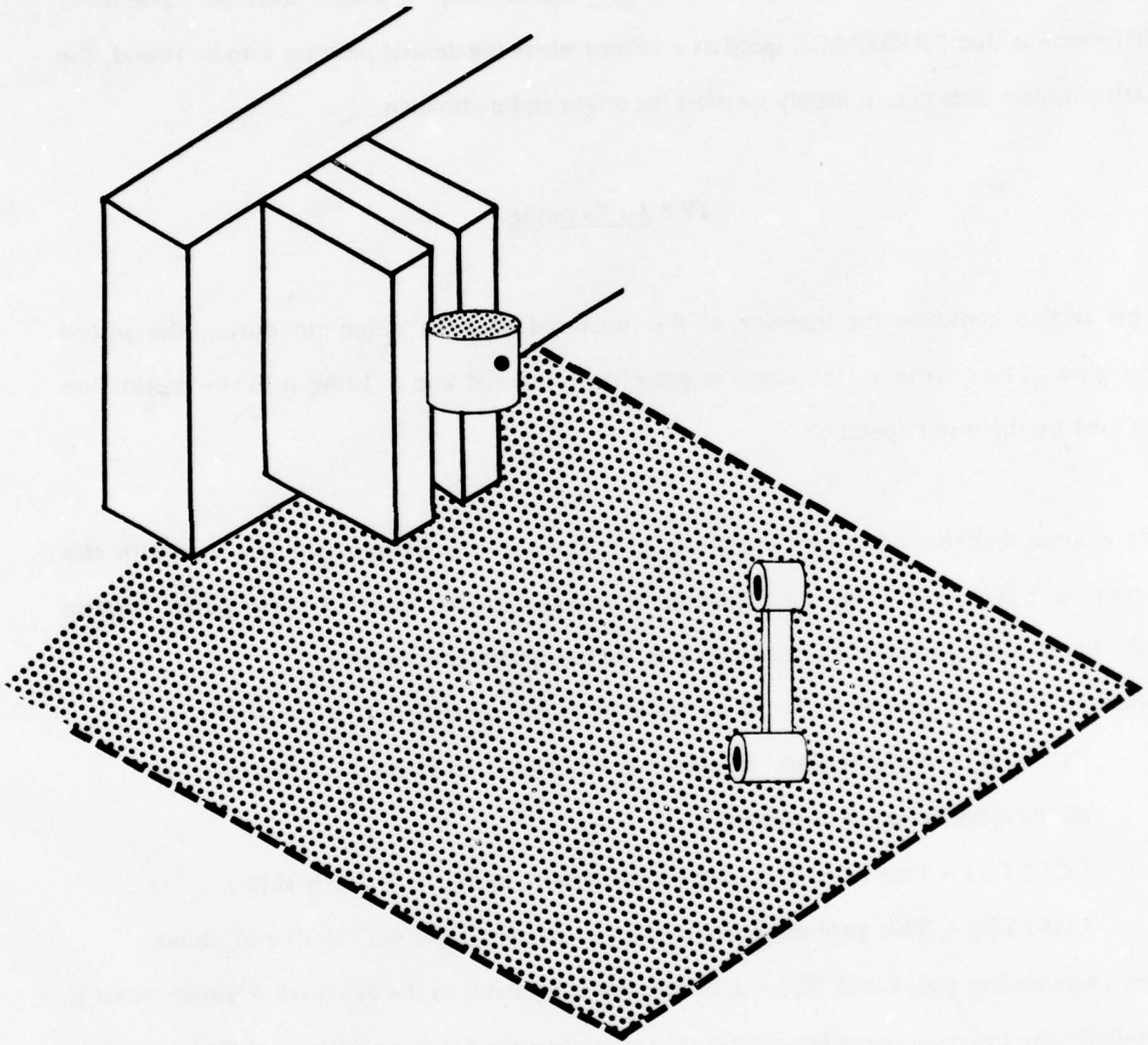


Figure 4.20 - Initial configuration for insertion of piston-rod on the piston-pin while it is held by the vise with the piston on it.

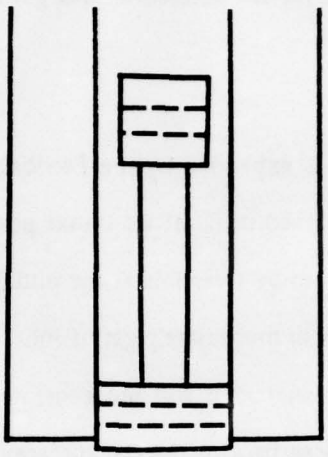


Figure 4.21 - LGSET[2] of rod's pin-end is accessible.

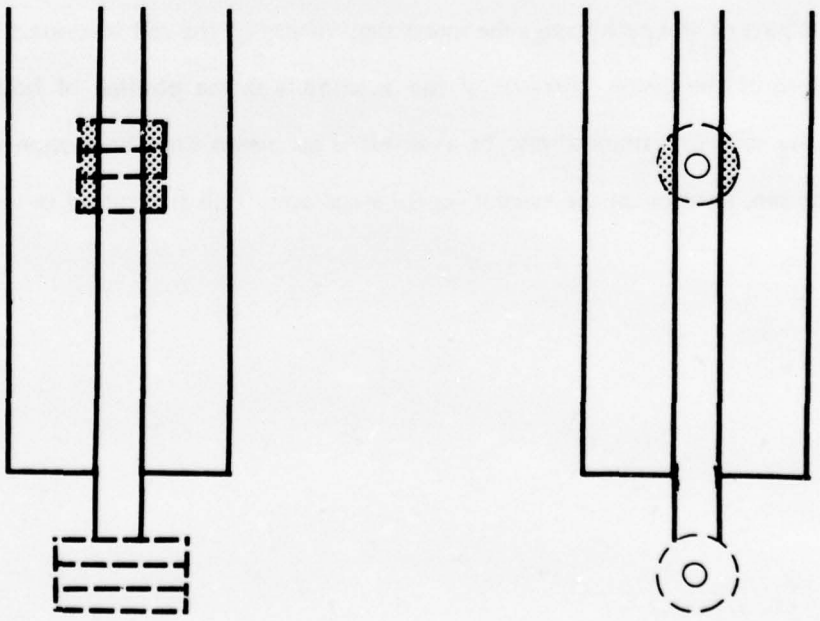


Figure 4.22 - LGSETs on rod's bar cause collisions.

The rod's shaft-end provides us with legal grasp points. The PGSET[1] is not accessible but both the LGSET[2] and LGSET[3] are accessible and generate no collisions at the initial position.

At this point the insert operation is expanded by the Feedback Planner. Three of the piston-rod's grasp sets were shown to be accessible at its initial position. Each motion is simulated using each of the grasp sets shown to be accessible at the initial position of the rod. In this case the grap set on the rod's pin-end will immediately get us into troule and will be ruled out. The LGSET[2] and LGSET[3] on the shaft-end will not generate any contacts and thus are both completely legal. LGSET[2] is chosen because the flat surfaces provide a better grip.

The expansion of the insertion operation results in a choice of a definite starting location. The next step is to compute a path to get there. Fig. 4.23 shows the simplest path. This path causes a collision with the piston. The collision is avoided by rising above the piston's height (Fig. 4.23). The last part of the path brings the interaction volume of the rod in contact with that of the pin and that of the piston. Because of the uncertainty in the position of both the piston and the pin, the contacts cannot always be avoided. This means that the motion to bring the rod next to the pin, must be on the lookout for these contacts. This is dicussed in section V.4.

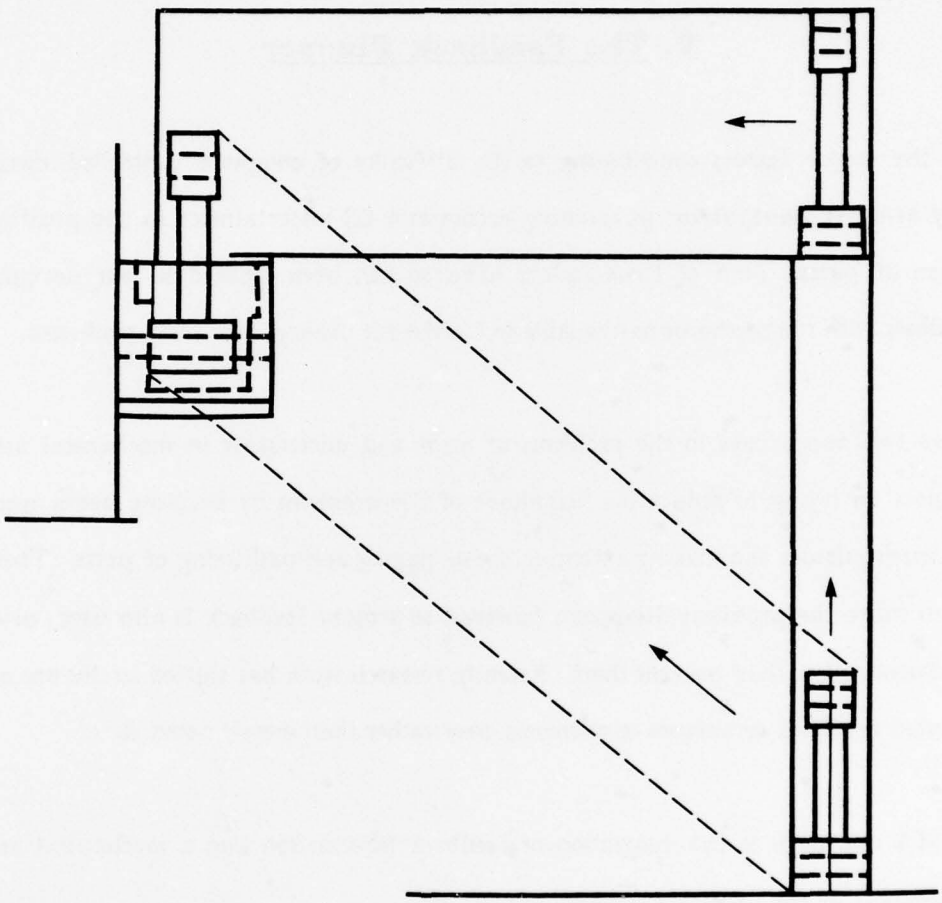


Figure 4.23 - Path from original position of piston-rod to that of piston-pin. The simplest path causes a collision with the piston, a path rising above the highest point on the piston is used instead.

V. The Feedback Planner

Two of the major factors contributing to the difficulty of computer controlled mechanical assembly are: (1) manipulator positioning errors and (2) uncertainties in the position and orientation of parts. Both of these factors have, so far, been ignored in our design. This chapter deals with the mechanisms available in LAMA for dealing with these problems.

There are two approaches to the problems of error and uncertainty in mechanical assembly. One focuses on trying to reduce the magnitude of the problems by building better and more accurate manipulators and making extensive use of jigging and palletizing of parts. This is not enough to make the problems disappear, however, so sensory feedback is also used, usually to detect failures rather than prevent them. Recently, research work has shifted to the use of more sophisticated feedback techniques to overcome error rather than merely detect it.

The LAMA approach to the integration of feedback information into a mechanical assembly systems stems from the following observations:

(1) The use of feedback information, such as force or touch, in an assembly task is best described as a program.

(2) Feedback programs are very difficult to write and/or debug.

(3) The details of an assembly program are often sensitive to the geometrical environment of the particular operation.

(4) The number of basic operations in assembly programs is quite limited.

The first two observations immediately suggest a program library of feedback programs for particular assembly tasks. The third observation suggests that any simple form of the library idea is infeasible. The dependence of feedback programs on geometrical context is crucial to the LAMA design. On the other hand, the fourth observation suggests that the geometrical information needed to modify the library programs can be embodied in an automatic system.

In Chapter I, I argued that Inoue's peg-in-hole insertion program must be modified to account for details of particular assembly steps. The operation of inserting the piston-rod onto the piston-pin requires knowing that the piston is also on the pin at the time (see Fig. 5.1). We must also consult the spatial relationship of parts to determine the effect of various failures during an assembly step. Knowing that if the piston-rod misses the piston-pin it will contact the inside of the piston suggests a check for the failure of that operation. No prepackaged program could use this information.

LAMA uses a library of *skeleton programs* to embody the structure common to all occurrences of the common assembly tasks, such as peg-in-hole insertion. The skeleton programs embody *assembly strategies* rather than assembly programs. These skeletons are specialized using the detailed knowledge of the environment present in the system's world model.

The *Feedback Planner* has the responsibility of expanding the assembly strategies into manipulator programs. Each strategy imposes constraints on the initial position of its arguments; these serve to define the range of initial positions and orientations for the objects. These ranges are used to compute the legal grasping positions. The strategy instantiation process places extra constraints on both the grasp positions and the initial position for the arguments of the strategy. After each strategy is expanded, the choice of grasp point and exact destination is made. Then, a collision-free path to that position can be computed.

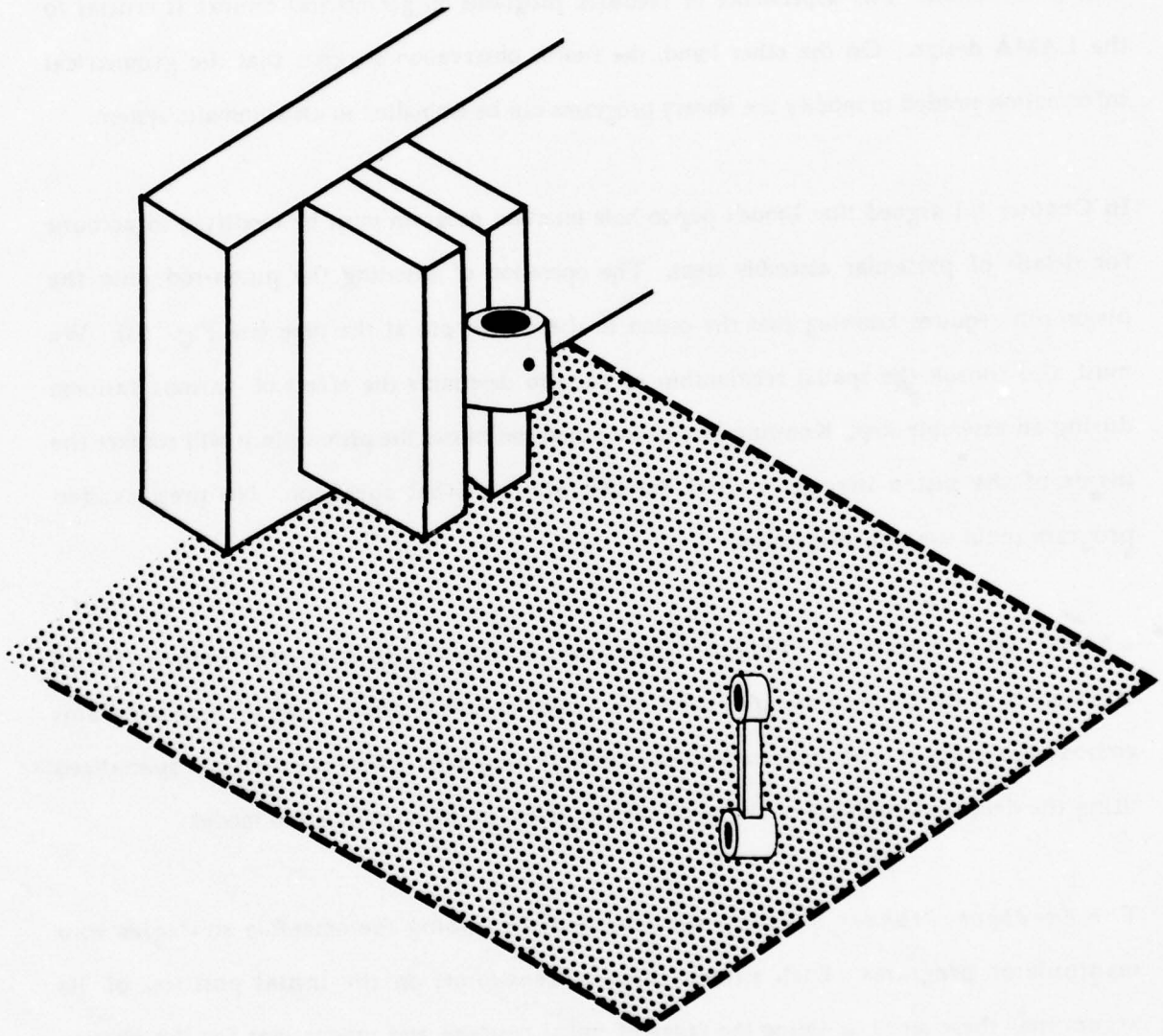


Figure 5.1 - Initial setup for piston-rod on pin insertion. See Fig. 4.21.

The instantiation process consists of a simulation of each statement in the assembly strategy with a view to predicting its possible results. The desired results of each operation are provided in the assembly strategy as constraints on the objects involved in the motion. This commentary and the corresponding simulation provide the information needed to specify any missing parameters of the motion commands. The simulation also allows the Feedback Planner to predict the possibility of failure and include extra code to check for these special occurrences.

The first section of this chapter deals with the assembly strategies and how they differ from manipulator programs. The other sections explain in some detail the operation of the Feedback Planner.

V.1 Assembly Strategies

An assembly strategy is a technique that uses the feedback information available to the manipulator (position, force, torque, touch, vision, etc.) to achieve a particular assembly state. An excellent example of an assembly strategy is Inoue's peg-in-hole insertion strategy [Inoue]. This section first considers Inoue's implementation of the strategy on the Little Robot System [Silver]. It then examines how such a program differs from the representation of the strategy in LAMA. The last sub-section is an overview of the task of the Feedback Planner: converting the assembly strategy back into a manipulator program.

V.1.1 A Manipulator Program

Fig. 5.2 shows a simple LISP program, taken from [Inoue], which implements the peg-in-hole insertion strategy. This program is made up out of five kinds of statements, collectively known as manipulator motions (or steps). Motions are specified independently for each of the manipulator's axes (x, y, z, r). These statements control the motion of the manipulator

either by specifying target positions or desired forces for any of the manipulator's axes.

(1) Force moves, e.g. (FX= force), indicates that the manipulator should move so as to maintain a force along that axis (x) equal to the specified force.

(2) Absolute position moves, e.g. (Z= position), indicates that the hand should be moved to a position having a coordinate value for the axis indicated (z in this case) equal to position. The other axes remain unchanged.

(3) Relative position moves, e.g. (DY= shift). Move the axis indicated by the amount of shift.

(4) Synchronization operations, e.g. (WAIT predicate), which means that the next step should not be executed until some predicate (an arbitrary LISP form) evaluates to non-NIL. Some special functions, e.g. (?FZ) or (?Y), have been provided which return T when the actual force or position on the axis is within some global value of the force destination, set by (FZ= force) or (DY= shift).

(5) LISP statements, e.g. (SETQ Z1 (GETMF ZPOS)). These programs are general LISP programs so any LISP operation can be executed. The special function GETMF is provided to access the values of the position and force parameters.

The program in Fig. 5.2 is a manipulator program that works on the particular configuration of parts used by Inoue in his assembly. The program must be slightly generalized if it is to serve as LAMA's knowledge of an assembly strategy. The next section describes how this is done.

V.1.2 Generalizing an Assembly Program into a Strategy

The program shown in Fig. 5.2 specifies the manipulator motions in terms of the manipulator's coordinate system. It also assumes, implicitely, that the motions of the hand correspond to the

```
(DEFUN DROP-INTO-C ( )
       (DR= 0.1) (DY= shift) (WAIT '(AND (?R) (?Y)))
       (FX= landing-force) (WAIT '(?FX))
       (DX= 0.0) )

(DEFUN MATE-H ( )
       (PROG (Z1)
             (FZ= small-edge-finding-force-to-"+z") (WAIT '(?FZ))
             (SETQ Z1 (GETMF ZPOS))
             (FZ= small-edge-finding-force-to-"-z") (WAIT '(?FZ))
             (Z= (//$ (+$ Z1 (GETMF ZPOS)) 2.0))
             (FY= small-contact-force-in-"y") (WAIT '(?FY))
             (FZ= 0) (R= 0.0) (WAIT '(?R)) ))

(DEFUN PUSH-INTO-H ( )
       (FY= 0) (FZ= 0)
       (FX= inserting-force) (WAIT '(?FX)) )
```

Figure 5.2 - Inoue's peg-in-hole insertion program.

motions of the *hole* where the insertion is to take place. More generally, an arbitrary transformation, involving displacements and rotations, is involved between motions of the hand and motions of a *feature* of the object in the hand. The first step in converting Inoue's program into an assembly strategy, is to describe the individual steps in terms of the motions of the desired features of the object in the hand. These motions should be specified relative to a coordinate system local to the objects.

Specifying the motions of parts relative to local coordinate systems instead of the motions of the manipulator in its local frame of reference achieves two important goals: (1) the motions are specified independently of how the objects are being held, or even of which object is being held; and (2) the motions are described independently of the manipulator.

Inoue's program does not do any explicit error checking. Any useful assembly program must detect failures and either correct them or abort the operation. Detecting failure requires that the desired effect of each of the manipulator steps be known. The desired final states can be described in terms of the same repertoire of geometric constraints we have seen used for object and assembly descriptions. Differences from these desired relationships can then be tested for. For example, the description of a force move can be expanded to include that the desired final state is the contact of two surfaces.

All the constants in a manipulator program have to be replaced by parameters, e.g. position of parts, force thresholds, shifts in positions, etc. These parameters are implicitely specified by the desired final state of the motion.

V.1.3 Defining an Assembly Strategy

An assembly strategy is the specification of the motions necessary to achieve a particular set of constraints on objects. Defining a strategy involves specifying the types of objects that can be operated on; the prerequisites on their position, orientations or degrees of freedom; the constraints on the objects established by the operation; the coordinate system used for specifying the motions; and the motion commands themselves. Fig. 5.3 shows the peg-in-hole strategy as represented in LAMA.

All motions are specified in a coordinate system defined relative to the arguments of the assembly strategy. The statements indicate motions of object features rather than of the hand. Motions changing only one of the position or orientation parameters of an object are said to involve only one axis. Motions affecting more than one of these parameters are said to involve more than one axis.

In principle, the transformation between the coordinate system to be used in the strategy and the manipulator's reference frame is arbitrary. The prototype system, on the other hand, restricts the assembly strategy to specify a coordinate system whose axes are parallel with the axes of the manipulator's coordinate system. This means that the transformation is generated by a displacement and rotations which are multiples of $\pi/2$ radians. This restriction is prompted by limitations of the Little Robot System. Arbitrary straight line motions are hard to define because the arm control software does not provide for coordination between axes. All the axes are independently servoed to their position or force destinations. Also, the construction of the force sensor complex and the fact that it is not uniformly calibrated across axes, preclude it being used to servo to arbitrary vector forces. These limitations are not too burdensome; most motions during assemblies involve only one of the manipulator's axes.

```
(STRATEGY PEG-IN-HOLE (PEG HOLE)
          (TYPE (PEG CYL) (HOLE CYL-HOLE))
          (REFERENCE (ALIGNED&CENTERED (REFERENCE X) (HOLE FRONT)))
          (PRE-REQS (CLEARANCE < 0.01))
          (INITIAL (AND (ALIGNED&CENTERED (PEG FRONT) (HOLE-FRONT))
                        (IN-FRONT-OF PEG HOLE)))
          (DROP : (DROP-INTO PEG HOLE)
           SUCH-THAT (PARTLY (FITS-IN PEG HOLE)))
          (MATE : (MATE PEG HOLE)
           SUCH-THAT (ALIGNED- PEG HOLE))
          (INSERT : (PUSH-INTO PEG HOLE)
           SUCH-THAT (FITS-IN PEG HOLE)))

(STRATEGY DROP-INTO (PEG HOLE)
          (ROTATE : (CHANGE R BY (RADIAN 0.1))
           SUCH-THAT (ALMOST (ALIGNED- PEG HOLE) (RADIANS 0.1)))
          (SHIFT : (CHANGE Y)
           SUCH-THAT (LEFT-OF (PEG CENTER) (HOLE CENTER)))
          (LANDING : (CHANGE X)
           SUCH-THAT (CONTACT (PEG FRONT) (HOLE FRONT))))

(STRATEGY MATE (PEG HOLE)
          (EDGE+ : (CHANGE Z)
           SUCH-THAT (AND (ABOVE (PEG CENTER) (HOLE CENTER))
                          (CONTACT PEG (HOLE SIDE))))
          (SAVE1 : (SETQ Z1 ZPOS))
          (EDGE- : (CHANGE Z)
           SUCH-THAT (AND (BELOW (PEG CENTER) (HOLE CENTER))
                          (CONTACT PEG (HOLE SIDE))))
          (SAVE2 : (SETQ Z2 ZPOS))
          (CENTER : (MOVE Z)
           SUCH-THAT (BETWEEN (PEG CENTER) Z1 Z2))
          (CONTACT : (CHANGE Y)
           SUCH-THAT (CONTACT PEG (HOLE SIDE)))
          (MATE : (CHANGE R WITH (AND (ZFORCE = 0.) (YFORCE = "MAINTAIN-CONTACT")))
           SUCH-THAT (ALIGNED- PEG HOLE)))

(STRATEGY PUSH-INTO (PEG HOLE)
          (PUSH : (CHANGE X WITH (AND (YFORCE = 0.) (ZFORCE = 0.)))
           SUCH-THAT (FITS-IN PEG HOLE)))
```

Figure 5.3 - The peg-in-hole insertion strategy.

The major types of statements found in assembly strategies are: motions of one axis, motions involving several axes, conditionals, loops, and procedure calls. The syntax is that of LISP with a few exceptions described in this section. More detail on the motion commands can be found in Appendix I.

The simplest and most common command is the single axis motion. The syntax is roughly:

(*stepname* : *motion* SUCH-THAT *constraints*)

stepname is merely a symbolic tag that can be used for transfer of control. The *motion* component of the statement specifies the axis to be moved and the details of the motion. The *constraints* indicate the relationship between the object features involved in the assembly strategy after the motion has been executed.

A *motion* command indicates either a position or a force destination, or both, for a manipulator axis. The axis is specified relative to the coordinate frame indicated for the assembly strategy. The position and force parameters of a motion statement can be specified either in "absolute" or in "relative" mode. Destinations can be described independent of the current value or as a desired change in value. Commands in relative mode are more frequent since they preserve information gained in previous motions.

In addition to the axis and the constraints, motion commands can supply the following information: (1) a force parameter, (2) a displacement parameter, (3) a force and a displacement, or (4) nothing. Each of the parameters can be either a desired end condition or a maximum allowable value; except that only one can be the desired condition, not both. The Feedback Planner must specify, numerically, both the force and displacement parameter for all motions.

A fully specified motion command completely describes the motion of the manipulator. The motion primitives used by Inoue affect the manipulator state exclusively by side-effects; so that

the effect of a motion command depends on previous commands. For example, if an axis is set to servo to a force, it will continue to do so until it is explicitely disabled. This makes the programs difficult to read and give rise to many errors when the programs are first being developed. LAMA's strategy language as well as its target language LLAMA (cf Appendix I), use the more self-contained syntax described above.

There are two kinds of motions involving more than one axis: (1) multiple independent motions and (2) multiple dependent motions.

Independent motions involve simultaneous changes to several position parameters of the object, so they generally involve path calculations. Multiple independent motion are specified by:

(SIMULT *motion motion* ...)

where each *motion* command is a single axis motion. I have avoided the use of multiple independent motions throughout the research. Their primary function is to speed up the execution of the manipulator programs; thus they provide an interesting optimization mechanism which should be focus of further research.

Dependent motions are either single or multiple axis motions that also entail some accomodation motions along other axes. An example of a multiple dependent motion is found in the MATE phase of the PEG-IN-HOLE strategy. The last command in the phase is to rotate the wrist so that the peg and hole are aligned. This involves a rotation (independent motion) while the z axis is servoed to zero force and a contact force is maintained along the y axis (dependent motions). This motion command can cause changes in the values of three of the position parameters of the object involved.

Conditionals test the manipulator's state after a motion and pass the control to different statements depending on the result. The syntax is that of LISP:

(COND *(test statements)* ...)

where the *statements* on each branch are either motion commands or transfer of control statements. Simulations involving conditional statements give rise to many problems [Taylor] [Rich and Shrobe] which I have not dealt with. The peg-in-hole strategy makes no use of explicit conditionals. This is not a coincidence; the most common use of conditional statements in manipulator programs is to detect (and correct) failure and since LAMA takes on most of the burden of detecting failure conditions, most of the conditional statements are generated by the system. In the current design, all error correction actions must be specified by the user. The user must guarantee that the corrective actions leave the environment in the same state (up to position uncertainty) as would succesful motions. Section V.3.3 discusses these problems in more detail.

Calls to other assembly strategies are handled as macro expansions. The instantiation process precludes the use of traditional procedure calls since the code of the strategy must be modified during instantiation.

Loops in assembly programs bring up many difficult problems, most of which I do not know how to handle. They will be totally overlooked until we consider the problems for further research in Chapter 7.

V.1.4 Fleshing Out the Skeleton

The goal of the Feedback Planner is to convert an assembly strategy into a manipulator program. The transformation involves specifying the motions in the manipulator's reference frame, picking numerical values for any parameters in the operations and inserting tests for likely errors.

The first task is to convert the specification for motions of particular features of the object in the hand into manipulator motions. For a manipulator whose degrees of freedom form a Cartesian coordinate system, e.g. the Little Robot [Silver], this can be done by a simple coordinate transformation. In fact, LAMA'S target language provides a mechanism by which all motion commands are interpreted relative to a local coordinate system. The REFERENCE statements in the assembly strategies serve to define the local coordinate system for the motion statements. Manipulators with cascaded rotational motions require more complex calculations e.g. see [Horn & Inoue].

One of the most difficult facets of the specialization process is adding the error detection information. Consider a force move meant to achieve contact between two surfaces. Two types of errors are possible under these circumstances: (1) The desired point of contact is missed or (2) contact is made elsewhere than expected. Each of the manipulator motions can be roughly simulated taking into account the uncertainties in the positions of the parts and the manipulator errors involved in the motion. This simulation can serve to identify sources of unexpected contacts and to place bounds on the positions from which the contact can still be achieved. These results can then be incorporated into the program. The command that specifies the force threshold to be met can also specify the position bound which indicates that the contact did not occur. We can also test whether the position of the contact (if it does ocurr) is in the range determined from the qualitative simulation.

Even when contact is made with the desired surface, errors may occur. The objects might slip on a curved surface or be jammed into a crack, etc. Detecting this type of error is much more difficult. In our scheme we would have to wait until the failure manifested itself by the failure of some other command.

Two important kinds of error conditions involve grasping. One is closing the fingers and finding nothing inside, indicating that either the part or the manipulator are not where they were expected to be. The other kind of error that must be detected is the slipping of objects in the manipulator jaws. This is fairly common since the parallel jaws constrain the motion of objects parallel to the grasping surfaces poorly. The strategies should be examined to determine if the reaction forces on the object are likely to cause slippage.

The parameters in the assembly strategy must also be specified and they generally depend on the particular parts involved. The legal values of `shift` in the `DROP-INTO` operation are different when inserting the piston on the pin than when inserting the piston-rod. The parameters in motions meant to establish constraints between parts can be deduced from the range of positions that will achieve the relationship, given the uncertainties in positions and the presence of other parts. Force thresholds can be determined from the nature of the actions involved. Only a few values are actually used, one to detect contacts, another to insert objects in tight fits, etc.

Most of the operations suggested above require a weak form of simulation of the manipulator motions. Enough simulation to be able to predict the possibility of some event occurring. This is so because the program must be on the lookout for information in order to be able to use it. The program cannot recognize a situation from its feedback information; it can only confirm it. Another purpose of the simulation is to constrain the motions well enough so that parameters can be specified for them.

V.2 Qualitative Simulation: Overview

May the Lord protect us from ghoulies and ghosties and things that go bump in the night.

This section describes the mechanisms available to the Feedback Planner for simulating assembly operations. The next section will go into more detail on how these tools are used to instantiate the assembly strategies.

The simulation process described here is both incomplete and qualitative. It is incomplete that it ignores many physical factors. It is qualitative because those factors that are considered are not treated quantitatively. We assume that motions are slow enough so that inertial effects can be safely ignored. We only consider the directions of forces and either ignore magnitudes or merely quantize them into a few values. In general, we attempt to reduce the detection of collisions to the detection of contacts. We simply detect the possibility of accidental motions rather than try to predict the properties and effects of the motions.

Simulation is currently in disrepute in AI circles so a few words are called for to justify its use here. The on-line analysis of feedback information, visual and/or force, for the purpose of detecting states of the world, is currently not feasible due to bandwidth limitations in the case of force and computational limitations in the case of vision. The alternatives to simulation are either to do careful experiments in a supervised environment or to analyze the assembly process. The complete lack of analytical or deductive tools to deal with 3-D space is amply demonstrated by one of the traditional "problems" of robot planning in the blocks world domain, FINDSPACE [Fahlman] [Sussman] [Pfister]. Because of the bandwidth limitation on force information, the experimentation alternative would either require human supervision or perfection of visual recognition tools. The former is undesirable and the latter is impossible in the short run.

V.2.1 Modeling Uncertainty

One of the key goals in our simulation of mechanical assembly operations is to model how they are affected by uncertainties in the positions and orientations of the objects involved. We would like a technique that did this without having to repeat the bulk of the simulation for different ranges in the values of positions and orientations. To a large extent this is unavoidable but we would like to limit its use as much as possible. The simplest way to do this is by treating the objects as if they were, simultaneously, at a whole range of positions and orientations. This technique has appeared earlier in the discussion of grasping, where the continuum of possible grasp points was treated explicitely.

When considering the representation of constraints on objects we introduced a representation of the position and orientation of the objects as a set of ranges of legal values of the position and orientation parameters. That was also a situation where we were representing degrees of freedom in the position and orientation of objects. We will adopt that representation throughout. The ranges of positions and orientations will be expressed as ranges relative to some *nominal position* which will be used when the position of the object needs to be known, e.g. when some other object is constrained relative to it. Similarly, the interaction volume we defined there can be adapted to our uses here. This volume serves to mark areas where entry of other objects might cause collisions. An example should clear this up.

Consider grasping a cylinder which is lying on the table and placing it in the vise. Its position is constrained to be within some small radius r of a known table position. Suppose that the orientation is known to be within the range ±45 degrees with respect to rotation around the manipulator's z axis (Fig. 5.4). This range of positions and orientations define the interaction volume of the cylinder. We have to open the fingers wide enough to insure that their volume does not intersect the interaction volume of the cylinder. This can be done by simulating

closing the fingers from their maximum width and detecting the first contact between the finger volume and the interaction volume of the cylinder. The required width to avoid the whole interaction volume might be more than the maximum opening of the fingers. In that case, a test for contact with the cylinder as the fingers approach the table must be included. The expected height of contact can also be computed from the simulation.

After grasping, the angular uncertainty of the cylinder's orientation is reduced to that of the hand. This is computed from the constraints generated by the grasping motion. These constraints also determine one of the degrees of freedom in the position of the cylinder's center. However, the center's position still has the same uncertainty range along the direction parallel to the grasping surfaces of the fingers. We can reduce this uncertainty by tapping a tip of the cylinder to an object at a known position. *Since there is only one degree of freedom in the position of the cylinder, one measurement is sufficient to determine it.*

The tapping operation requires predicting where the contact will occur. The positional information obtained from a tap depends on the relation between the point of contact, the known position and the manipulator's position. Merely tapping, without knowing where the tap point is, is worthless. The ability to predict where contacts will occur is crucial to the Feedback Planner. We will discuss it in the next sub-section.

The example above dealt with a single object, subject to a single source of uncertainty. This is the simplest case. The more general situation requires computing the result of uncertainty relative to an uncertain frame of reference which is in turn uncertain ... An example is the position of an object in the fingers of a manipulator. There, the uncertainty in the position of the object with respect to the fingers is compounded by the uncertainty in the absolute position of the fingers. This type of *cascading* error can become disastrous for multiply jointed manipulators, where each joint introduces an angular error which is magnified by the length of

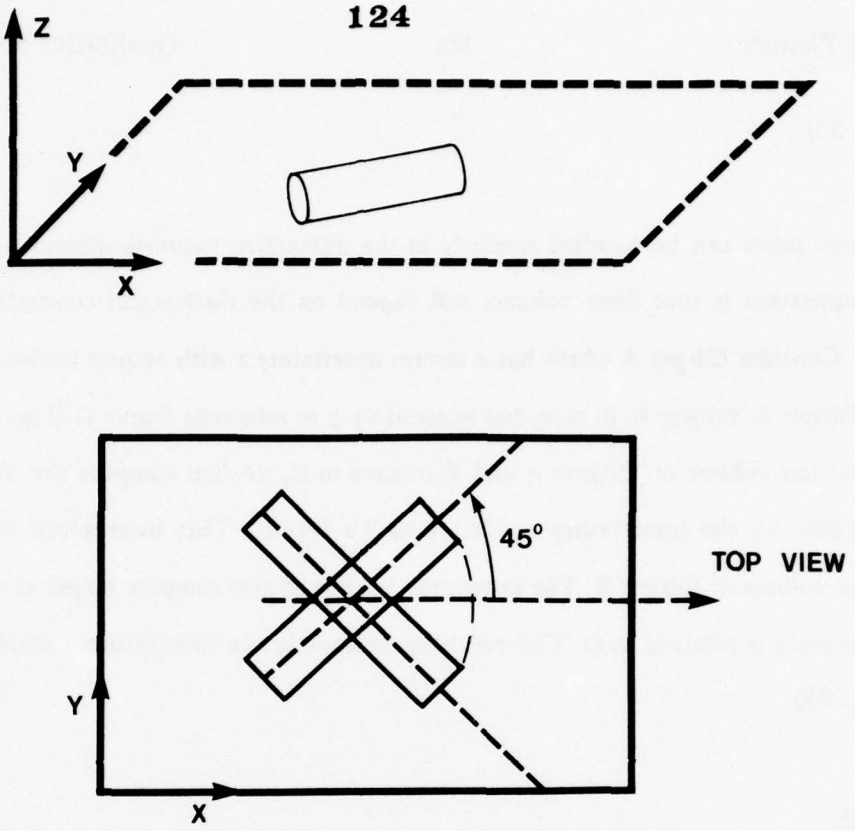


Figure 5.4 - The pin's position and orientation are uncertain. Grasping requires computing how wide the fingers should open to avoid collisions with the volume generated over the range of uncertain positions.

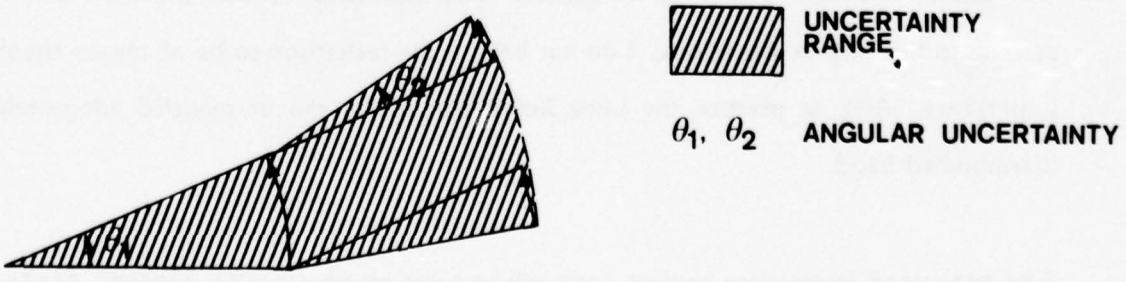


Figure 5.5 - Area generated by cascaded angular error.

the link (Fig. 5.5).

Cascaded uncertainties can be handled similarly to the interaction volumes mentioned earlier. The only complication is that these volumes will depend on the mechanical connections with other objects. Consider Object A which has a certain uncertainty x with respect to the frame of reference of Object B. Object B, in turn, has uncertainty y in reference frame C (Fig. 5.6). To find the interaction volume of Objects A and B relative to C, we first compute the interaction volume generated by the uncertainty x relative to B's frame. This interaction volume is attached to the volume of Object B. The interaction volume of this complex object is computed for the uncertainty y relative to C. The resulting volume is the interaction volume of the structure (Fig. 5.7).

V.2.2 Contacts

The most important operation in the qualitative simulation of assembly operations is detecting contacts between interaction volumes. The details of the volume intersection procedure are presented in Appendix 2. This section discusses how these intersections are used.

Throughout this section I will continue to make use of the "disembodied hand assumption", that is; I will model the manipulator as a hand composed of a large, cuboid wrist and two fingers. The rest of the manipulator will be ignored. This assumption reduces the work load on the system and on my explanations; I do not believe the restriction to be of major theoretical importance. Also, in practice, the Little Robot manipulator can be modeled adequately as a disembodied hand.

The analysis of interactions between parts will be based on what I call a *contact history* for each manipulator motion. This history goes through several stages. The first stage is that of a

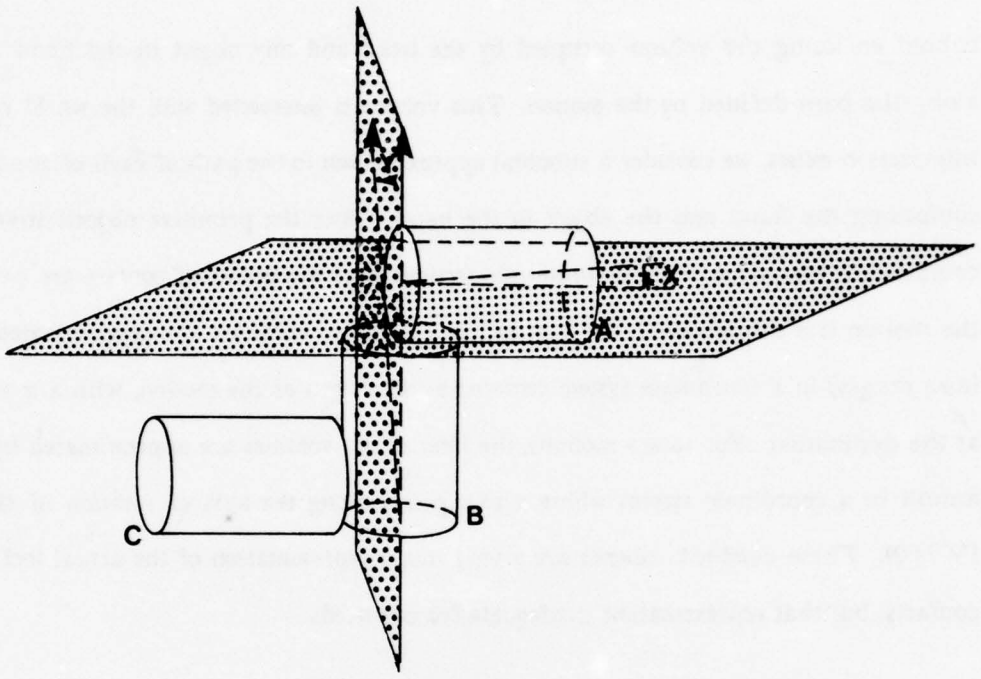


Figure 5.6 - Example for computing uncertainty volume for cascaded uncertainty. x indicates the angular uncertainty of the axis of A with respect to the axis of B and y is the uncertainty of B's axis w.r.t. to C's.

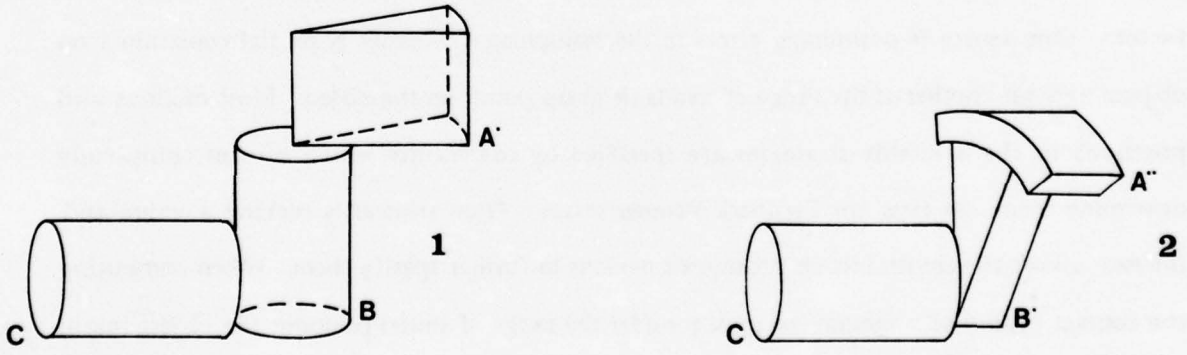


Figure 5.7 - Computing uncertainty volume for cascaded uncertainty:
step 1. Compute an approximation to the volume generated by the outermost object over its range of positions.
step 2. Assume that the volume generated by step 1 is part of the object and repeat for next link.

cuboid enclosing the volume occupied by the hand and any object in the hand as it travels along the path defined by the motion. This volume is intersected with the world model. If an intersection exists, we consider a cuboidal approximation to the path of each of the basic objects composing the hand and the object in the hand. Once the primitive objects involved in the contacts are identified, more accurate intersections near the points of contact are performed. If the motion is a displacement, the intersection volumes so obtained are approximated by cuboids (*xyz* ranges) in a coordinate system centered at the origin of the motion, with a *z* axis pointing at the destination. For rotary motions, the intersection volumes are approximated by sections of annuli in a coordinate system whose *z* axis points along the axis of rotation of the hand (cf. IV.7.1.2). These *contact ranges* are a very rough representation of the actual loci of potential contacts, but that representation is adequate for our needs.

The contact history is computed, not merely for the volume of the object and the hand, but for their interaction volume. This interaction volume is generated jointly by the object's volume and by its "degrees of freedom" (cf. III.3.1). These "degrees of freedom" stem from three factors. One source is positioning errors in the manipulator; another is partial constraints on objects and yet another is the range of available grasp points on the object. Most motions and positions in the assembly strategies are specified by constraints which do not completely determine them. At first, the Feedback Planner refrains from arbitrarily picking a value and, instead, allows the constraints on subsequent motions to further specify them. When computing the contact history of a motion, we must consider the range of initial positions the object might have and the range of grasping points as well as the uncertainty in those positions.

Each motion has a set of constraints it is meant to achieve. Included in these constraints might be some CONTACT relationships. These are the "desired contacts" of the motion; all other contacts are "accidental contacts". The processing differs slightly for these two types of contacts.

There are several pieces of information we want from a contact history; we will list them before discussing them in more detail.

(1) What is the range of initial positions for the motion that will achieve the desired geometrical constraints betwen the parts and avoid most accidental contacts?

(2) What restrictions on grasping the object are imposed by the motion?

(3) For accidental contacts, that cannot be always avoided by (1), is there any situation where the contacts do not happen?

(4) Divide the contacts in (3) into ambiguous and unambiguous contacts. An ambiguous contact is one whose range of values in the displacement parameter of a motion overlaps that of another potential contact. For such contacts the end position of the manipulator does not uniquely identify the object encountered.

(5) Specifiy the earliest and latest point of possible contact for each potential contact, desirable or accidental.

This provides all the information which the Feedback Planner needs to fully specify a motion and to plan for any contingencies. Section V.3.1 dicusses how this information is used. The rest of this section discusses the contact history computation used in the current implementation. The implementation and therefore the discussion is limited to single axis motions.

A motion in a LAMA strategy is described mainly by the constraints it is meant to achieve; this description is transformed by the process described in Chapter 3 into ranges of positions and orientations relative to a base coordinate system. When simulating a motion we know the range of initial positions of the hand (obtained from the previous motion) and we can compute the range of desired final positions from the constraints. The intersection of these two ranges determine the range of initial positions for the motion that will achieve the motion's goals.

We can take advantage of any remaining degrees of freedom in the initial position to avoid accidental contacts. Each of the accidental contacts is described as a range of positions of the manipulator which might cause a collision. These ranges can be removed from the range of initial positions for the motion. The contacts are first ordered as follows: (1) contacts that might give rise to motions (cf V.2.3), (2) ambiguous contacts, i.e. where the contact might be due to different objects and (3) other contacts, ordered by their size. If any contact would reduce the initial range to the null set, it is placed on a list of ***unavoidable contacts***.

Unavoidable contacts are collisions that cannot be completely ruled out by choosing where to start a motion. The fact that we take into account uncertainty in position and orientation as well as ranges of grasp points when computing the loci of potential contacts means that these unavoidable contacts might, in fact, not always happen. We can determine whether these collisions always happen or not by checking if there are positions within the degrees of freedom of the colliding objects where the contacts do not occur. The grasp computation described in Chapter 4 can be used to determine if the collisions with the volume generated by the hand ranging over a grasp set can be avoided. If a contact can be avoided in this way, then it is removed from the list of unavoidable contacts and the set of legal grasp points for the motion is restricted appropriately. These are called the grasp constraints on the motion. The uncertainty in the position of the objects should also be examined to determine if the collisions can usually be avoided. If so, we include a test for the collision in the manipulator program. If the collision is really unavoidable, then the operation is impossible. The current implementation bypasses this step and assumes that contacts are never completely unavoidable.

The unavoidable contacts and the desired contacts are all represented as ranges of values in the manipulator's coordinate system. These ranges are used to fill in any missing parameters of the motion and to interpret the end state of the motion when it is actually carried out.

V.2.3 Predicting Motions

The ability to predict "unscheduled" motions is important to the Feedback Planner. Consider the situation after the piston-rod has been inserted onto the pin inside the piston (Fig 5.8). Releasing the piston-rod could cause it to rotate on the pin under the influence of gravity. The rotation would then bring the rod in contact with the edge of the cylinder. This might rotate the piston. The system's model cannot predict what will happen but, since the next step involves grasping the piston, it needs to know. This section describes how this kind of consideration could be incorporated into the process of instantiating assembly strategies. As of this writing this portion of the system design has not been implemented.

For each manipulator motion two possibilities must be investigated: (1) Were there forces exerted during the operation? (2) Were any motions enabled by the operation? If the answer to either question is affirmative, we consider whether the objects involved are free to move in the direction of the force (and reaction). If any are, we must determine whether this new motion might cause any further motion. This is done by computing the collision history of the motion. The process repeats for any new collisions detected. This recursive process generates a tree of possible motions. Any motions, precluded by constraints in the assembly plan, are not pursued. The user can then be asked which of these motions are likely to happen. Tne results are incorporated to the uncertainty in the position and orientation of the objects. In general this process might be combinatorially explosive but I believe that, in practice, it is feasible.

Simulation can be used to determine if an object is free to move in a particular direction. This is a computation similar to the grasping process described in Chapter 4. In both cases we have objects that can be located at a range of positions and we are interested in determining which subrange of positions are illegal because of the presence of other objects in the environment. The analogy is even stronger since the type of motions we are usually interested in are either

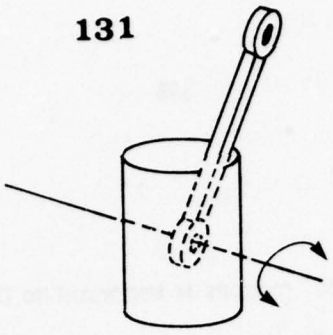
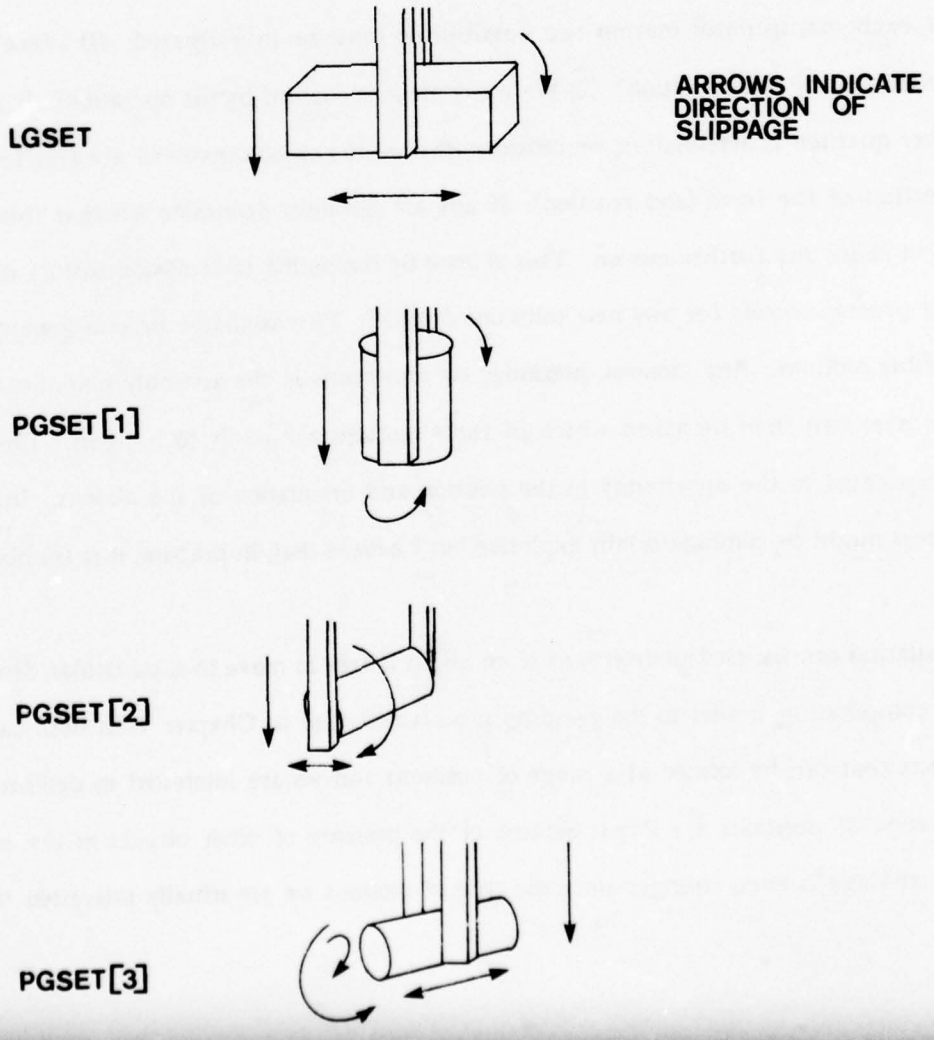


Figure 5.8 - Possible motion of rod rotating on pin inside piston. The sides of the piston constrain the motion.

Figure 5.9 - Slippage conditions for grasp sets. The arrows indicate possible directions of motion.



constrained to lie on a plane (LGSET) or to rotate around a fixed axis (PGSET). For example, the rod on the piston-pin has a legal range of positions which can be simply described in terms of a cylindrical coordinate system centered on the pin. The range of positions that can be reached by moving without causing a contact can be computed similarly to the way the legal grasping positions are obtained.

One of the most important uses of the motion computation is to detect the possibility that the reaction forces generated during an assembly operation might cause the object in the hand to slip. This computation depends on the particular grasp position and the maximum forces generated in the motion. All grasping positions (GSETS) have slippage conditions (Fig. 5.9). Again, the user can be asked to estimate how likely the motion is, since it depends on many factors beyond the scope of the Feedback Planner e.g. friction, regularity and detailed shape of the grasped object.

V.3 Instantiating the Assembly Strategies

Instantiating an assembly strategy involves expanding each of the statements in the strategy so as to fully specify its arguments and to provide for any contingencies during its execution. This section discusses the application of qualitative simulation to this instantiation process. Section V.4 presents an example. The current implementation is restricted to single axis motions and therefore our discussion in this section will be limited to those motions.

V.3.1 Processing the Motion Commands

The basic conceptual unit of an assembly is a GRASP -> OPERATION -> UNGRASP sequence, where OPERATION is one or more calls to assembly strategies. Examining the assembly plan in Fig. 1.8 shows that the piston assembly is composed of four of these sequences. This section

describes the processing of these basic operations. We will ignore the possibility of accidental motions, not because they are unimportant but because the current implementation does not handle them.

The first step in the processing is determining the legal ways to grasp the object at its initial position. This will identify a few reachable grasp sets to which we can confine our attention. Then, for each motion command in the assembly strategies, we must compute the contact history and the grasp constraints for each reachable grasp set. This generates the following information for each motion: (1) a range of legal initial (and final) positions, (2) a set of legal grasp positions for each of the reachable grasp sets of each motion, (3) the ranges of manipulator positions for the desired contacts and (4) ranges of manipulator positions for possible accidental contacts.

The information for each motion must then be integrated for the whole operation. This is done by a relaxation procedure involving the initial and final ranges of each motion and an intersection of the legal grasp positions for each reachable grasp set.

Each motion's range of initial positions is inherited from the immediately preceding motion's range of final positions. This initial range is further constrained by the requirements of the particular motion. This process continually constrains the initial and final range for the motions as the simulation progresses. When all the motions have been considered we must work backwards to propagate the constraints by intersecting the initial range of each motion with the final range of the previous motion. After this is done, an arbitrary choice of starting point can be made from the initial range of the first motion and the choice can be propagated without fear that the arbitrary choice will make the operation impossible.

We also computed the range of legal grasp positions for each reachable grasp set of each motion command. We can now choose a definite grasp position by intersecting all the ranges of positions for each motion together with the original range of grasp positions for the object in its original position. We can pick a grasp position from this final range.

Once we know where to grasp the object we can compute a path from the hand's initial position to the grasp point and from there to the initial position for the operation. The last operation to be considered is the UNGRASP which is trivial given that we are not considering the unexpected motions that it might give rise to.

V.3.2 Code Generation

The next step in the process of instantiating the assembly strategies is to generate the code for the motion commands. Each motion command generates two LLAMA statements: (1) a fully specified motion command (cf Appendix 1) and (2) a conditional statement that examines the results of the motion and decides if the motion was succesful.

There are two basic types of single axis motions; those that have a contact between two objects as part of their goal and those that do not. Both types of motion generate the same type of LLAMA motion command:

(CHANGE *axis* BY *"distance"* WHILE *(force < "detect-contact"))*

This command indicates that the specified axis is to be moved until either a distance greater *"distance"* is covered or the force threshold is exceeded. This defines the two normal termination conditions for a motion POSITION and FORCE. There is a third termination condition, TIME, when a motion is aborted after a time greater than a global threshold elapses without perceptible motion. The TIME termination provides a fail-safe condition for situation in which the contact force does not exceed the threshold. Both the FORCE and TIME termination

indicate a contact occurred.

The "*distance*" parameter for motions with desired contacts is chosen to be the upper bound on the contact range for the desired contact, this is obtained from the contact history. For other motions, a choice is made from the range of displacements that will achieve the desired results.

The force parameter is equally easy to specify. If a contact, desired or accidental, might happen, the global threshold "*detect-contact*" is used. Otherwise, a higher threshold called "*emergency-stop*", is used. These thresholds can be adapted to the particular arm or by the nature of the objects being dealt with by the user.

The second part of the code generated for a single axis motion tests the termination condition of the most recent motion and examines the end position to determine the result of the motion. The basic function used for this purpose is CONTACT?:

(CONTACT? *axis min max*)

is true if the termination condition is FORCE or TIME and if the position value of the specified manipulator axis lies between *min* and *max*. The conditional statements are made up of three type of branches:

(1) testing for desired contacts: ((CONTACT? *axis min max*))

(2) testing for accidental contacts: ((CONTACT? *axis min max*) (ERROR))

(3) default error condition for motions with desired contacts: (T (ERROR))

Section V.4 will show several examples of how this mechanism is used. Whenever a possible error is detected the user is informed of it and given three choices: (1) declare the error unlikely and have it be ignored, (2) declare it fatal and have it generate an error or (3) specify a corrective action. Corrective actions currently have to be specified in the strategy language.

V.3.3 Flow of Control

This section discusses the problems with simulating conditional statements and outlines a limited approach to the problem. This approach has not been implemented. The current implementation only deals with assembly strategies with straight-line code, i.e. no loops or conditionals.

Conditionals are of two types: (1) merging and (2) non-merging. This division depends on whether the control paths leading out of the conditional ever merge. Conditional statements that have merging control paths are difficult to simulate. The positions and orientations of objects might be different depending on which of the paths is taken. The Feedback Planner cannot decide a-priori which set of positions to use in the statements following the merge of the control paths.

Consider the insertion of the rod onto the piston-pin while the pin is inside the piston. Because of errors in grasping the piston-pin, there might not be enough room for the piston-rod between the tip of the pin and the inside wall of the piston. Since the piston is free to move on the pin, we might push the piston back along the pin until there is enough room. Then the insertion can be carried out. The problem is that now the position of the piston on the pin cannot be predicted very well before execution; but we need to know its position when we want to take it out of the vise. In this case, the errors introduced are fairly small and will not seriously affect the rest of the assembly. This need not be the case.

We can adopt the following restriction on merging conditionals in assembly strategies: The differences in the positional information between the branches of the conditional, can be combined into a continuous uncertainty range for each object. Thus the result of the conditional is to increase the uncertainty in the position of objects. The operations following

the merge must work under that uncertainty. This was the case in the example above.

The conditionals introduced by the error predictions considered in the previous sections are by themselves non-merging since they generate error conditions. Allowing the user to specify actions to be undertaken when the errors are detected makes them into merging conditionals. The current design calls for allowing the user to specify corrective actions and assuming that the control paths will merge correctly and that the simulation is not affected by which path is taken.

V.4 The Feedback Planner: A Scenario

This section considers the operation of the Feedback Planner during the expansion of the PEG-IN-HOLE operation in which the piston-rod is to be inserted onto the piston-pin, while the pin is inside the piston (Fig. 5.1). A preliminary implementation currently exists of the program that computes the contact histories and does the simple code generation shown here.

The assembly plan (Fig. 1.8) has the following entries for the operation of inserting the piston-rod on the piston-pin:

```
(GRASP OBJ : [ROD]
       SUCH-THAT : (FACING ([ROD BAR] TOP) UP))

(INSERT OBJ1 : [PIN]
        OBJ2 : [ROD PIN-END-HOLE])

(UNGRASP OBJ : [ROD])
```

The plan specifies that the rod be upright. This could have been determined by an Assembly Planner from the requirements of the insert operation, but we will assume the user specified it. We must first find a grasp point on the piston-rod. This is done in the fashion described in Chapter 4. There are two possible grasp positions on the rod (Fig. 5.10); one along the sides of the piston-rod's pin-end, the other on the flat ends. The choice will depend on several factors: (1) flat surfaces are preferred to curved surfaces, (2) possible collisions and (3) which grasp position can better withstand the reaction forces generated during the assembly operation. The current implementation only considers the first two factors and since neither grasp set produces collisions, grasping along the flat ends will win out. One way to determine this is for the Feedback Planner to instantiate the INSERT independently for each grasp set. If one grasp position produces less possibility of error than the other, then it would be chosen. This method of handling multiple grasp position is wasteful because many of the same operations are done for each grasp position. The alternative currently used is to carry forward as if the hand position were uncertain. The Feedback Planner can then consider the effects on each of the grasp points simultaneously.

The first task in expanding an assembly strategy is to setup the local reference system. The REFERENCE statement in DROP-INTO indicates that the reference frame's x axis is ALIGNED&CENTERED with the HOLE's front face. This leaves one rotational degree of freedom unspecified. The current system always tries to line up unspecified degrees of freedom in the reference with global axes. In this case, the reference's z is aligned to the global z.

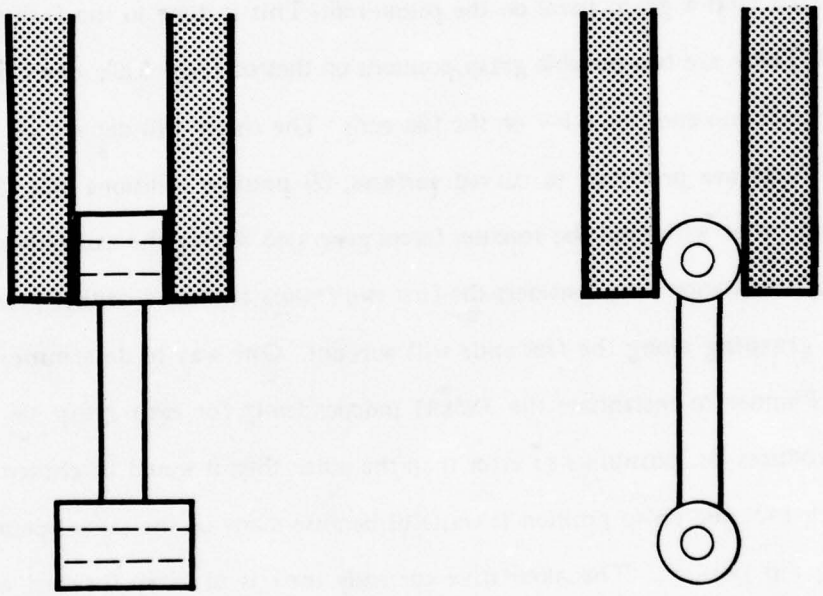


Figure 5.10 - Grasp positions on the piston-rod's shaft end.

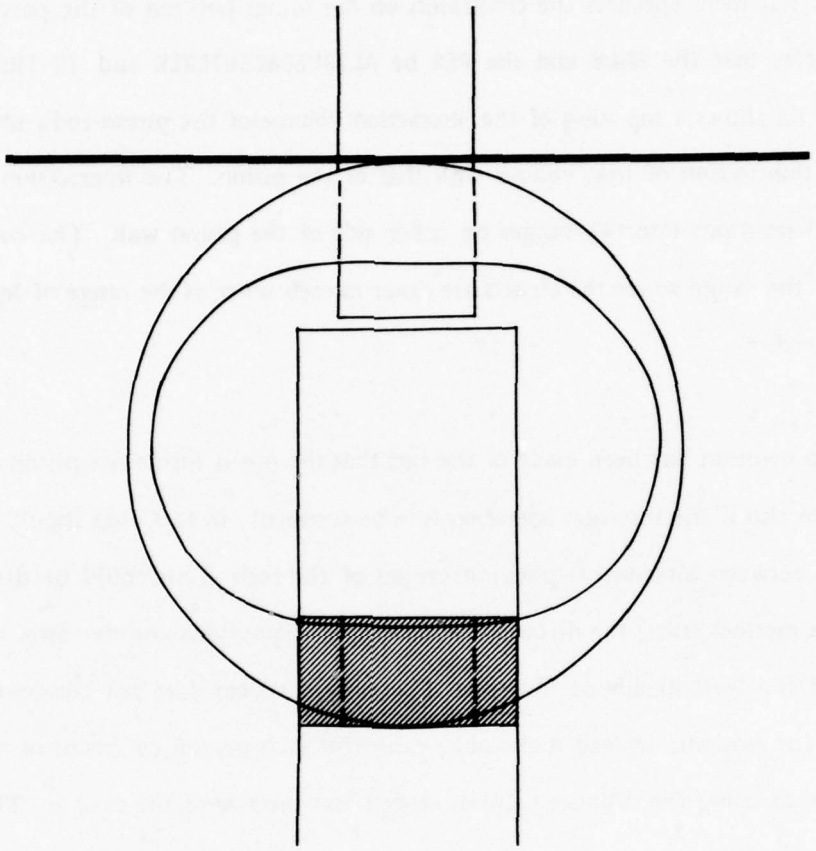
The INITIAL statement specifies the constraints on the initial position of the parts. In DROP-INTO it specifies that the HOLE and the PEG be ALIGNED&CENTERED and IN-FRONT-OF each other. Fig. 5.11a shows a top view of the interaction volume of the piston-rod's small end and indicates the intersection of that volume with that of the piston. The intersection divides the range of legal positions into two ranges on either side of the piston wall. The current system chooses to use the range where the objects are closer to each other as the range of legal positions of the piston-rod.

Notice that no mention has been made of the fact that the pin is inside the piston and the rod has to be there also if the insertion operation is to be succesful. In fact, that should be the basis for choosing between alternative position ranges of the rod. This could be discovered by simulating the motions using the different initial ranges of positions and deciding on the range that produced the least likelihood of error. The current system does not consider alternative initial ranges for motions; instead it chooses a range for each motion by means of a few simple heuristics such as using the distance between objects and the size of the ranges. This situation should be modified in other implementations.

The first step in the DROP-INTO strategy calls for the object in the hand to be rotated 0.1 radians. The Feedback Planner must establish that this rotation will not have any deleterious effects. This is done by computing the collision history of the motion. In this case, contacts with the pin and/or the piston are possible. These accidental contacts determine that the force parameter be *"detect-contact"* and that an error should be generated if the termination condition indicates a contact. The code that does this is shown here:

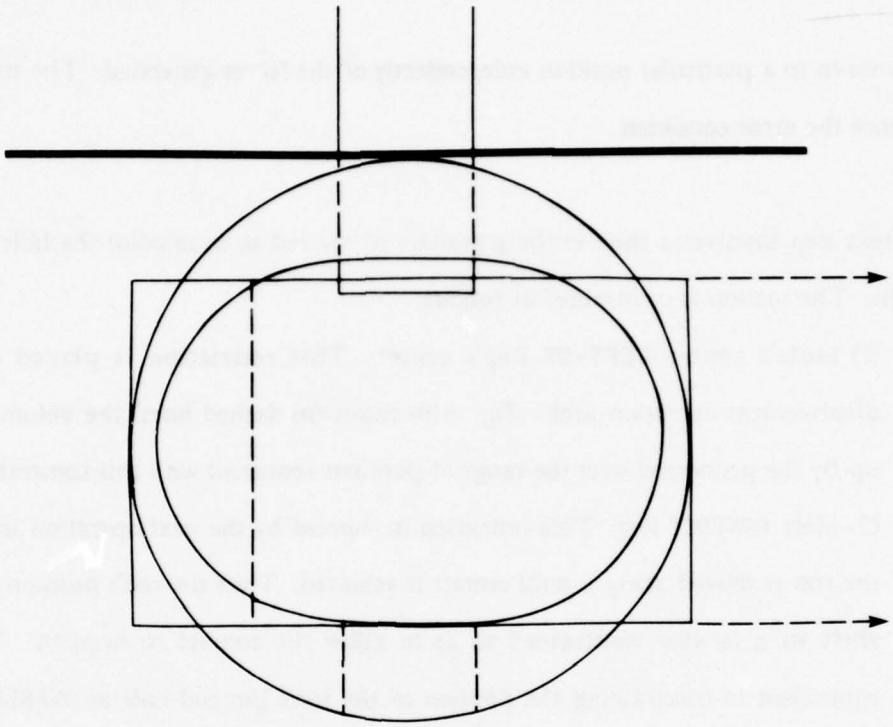
```
(CHANGE R BY (RADIAN 0.1) WHILE (RFORCE < "detect-contact"))
(COND ((CONTACT? R 0.0 0.1) (ERROR)))
```

At this point the user is asked about the likelihood and seriousness of the predicted error. A collision detected at this point in the assembly is not really very serious. The important thing is

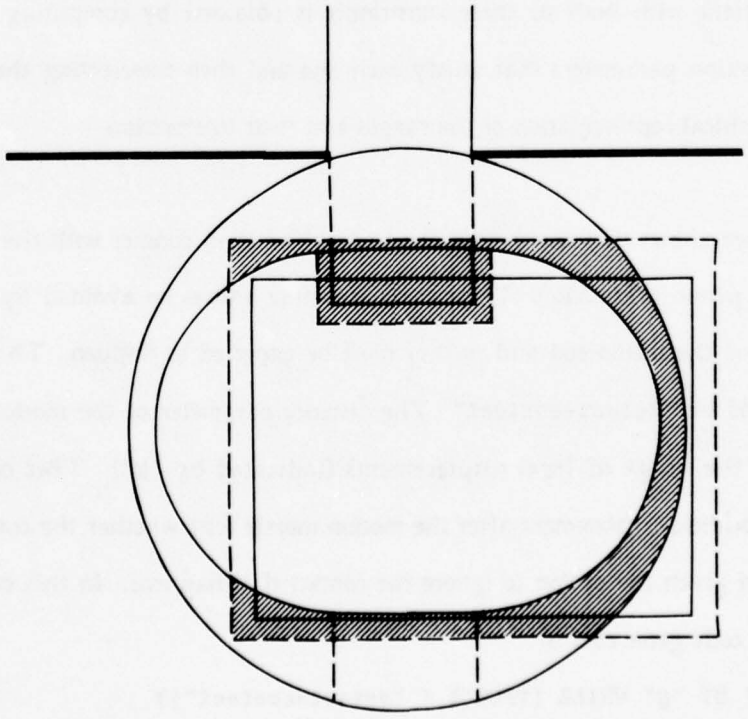


a

Figure 5.11 - The ranges of positions in the DROP-INTO operation (top view into the piston cavity):
(a) Volume taken up by piston rod over the range of positions satisfying:
(ALIGNED&CENTERED (PEG FRONT) (HOLE FRONT))
and
(IN-FRONT-OF PEG HOLE))
The shaded area indicates potential contact with the piston wall.
(b) The dashed area indicates the rod's volume over the range of positions consistent with:
(LEFT-OF (PEG CENTER) (HOLE CENTER))
The solid lines indicate those positions consistent with:
(CONTACT (PEG FRONT) (HOLE FRONT))
The intersection is shown lightly shaded.
(c) The potential contacts in shift when uncertainty is taken into account.



b



c

not to servo to a particular position independently of the forces generated. The user can choose to ignore the error condition.

The next step involves a shift in the *y* position of the rod so as to place the hole to the left of the pin. The motion is constrained as follows:

(1) Hole's center LEFT-OF Peg's center: This restriction is placed on the displacement operation itself. Fig. 5.11b shows (in dashed lines) the volume taken up by the piston-rod over the range of positions consistent with this constraint.

(2) Hole CONTACT Peg: This restriction is imposed by the next operation in which the rod is moved along *x* until contact is achieved. Thus the rod's position for the shift in *y* is also constrained so as to allow the contact to happen. This is equivalent to constraining the position of the rod's pin-end-hole to OVERLAP in *y* that of the piston-pin's front face. Fig. 5.11b shows (in solid lines) the volume of the rod over the range of positions consistent with this constraint.

A position consistent with both of these constraints is obtained by computing the range of values of the position parameters that satisfy each one and then intersecting the ranges. Fig. 5.11b shows a graphical representation of the ranges and their intersection.

The contact history shows that the motion can bring the rod in contact with the pin and with the inside of the piston (Fig. 5.11c). These contacts cannot always be avoided by adjusting the starting position of the piston-rod and so they must be expected to happen. This dictates that the force threshold be "*detect-contact*". The distance parameter of the motion is chosen as the midpoint of the range of legal displacements (indicated by "*y*"). This choice is quite arbitrary. The conditional statement after the motion merely tests whether the contact occurred. The user is again given the option to ignore the contact if it happens. In this case that is the best course. The code generated is:

```
(CHANGE Y BY "y" WHILE (YFORCE < "detect-contact"))
```

```
(COND ((CONTACT? Y 0.0 "y") (ERROR)))
```

After the shift operation, the contact history for the landing step is considered. A contact can always be achieved, but there is a region of uncertainty where contact with the inside of the piston is possible before contact with the pin. The contact is ambiguous, so the error cannot be detected by using the location of the contact. The code generated simply makes sure that the contact is in fact detected. The displacement used in the motion is the displacement necessary to go past the last possible contact with the piston-pin and collide unambiguously with the piston wall.

```
(CHANGE X BY "x" WHILE (XFORCE < "detect-contact"))
(COND ((CONTACT? X 0.0 "x")) (T (ERROR)))
```

This completes the DROP-INTO operation. The complete LLAMA program can be seen in Fig. 5.12.

The next step is to compute a path from the position where the piston-rod is first grasped to that where the INSERT is to happen. Fig. 5.11a shows the position chosen for the initial position of the piston-rod. The details of the path computation for this example were shown in Chapter 4. A straight line path to this position is not possible since it implies going through the piston. The collision avoidance routine generates a path that goes above the piston and moves down to the desired position. In this path the possibility of a collision cannot be removed completely, because of the uncertainty in the position of the pin, the piston and the tip of the rod. The collision avoidance routine passes the contact history to the Feedback Planner which generates the code for this motion. A schematic of the situation is shown in Fig. 5.13.

```
(STRATEGY DROP-INTO (PEG HOLE)
          (ROTATE : (CHANGE R BY (RADIAN 0.1))
           SUCH-THAT (ALMOST (ALIGNED- PEG HOLE) (RADIANS 0.1)))
          (SHIFT : (CHANGE Y)
           SUCH-THAT (LEFT-OF (PEG CENTER) (HOLE CENTER)))
          (LANDING : (CHANGE X)
           SUCH-THAT (CONTACT (PEG FRONT) (HOLE FRONT))))
```

```
(DEFUN DROP-INTO (PEG HOLE)
     (CHANGE R BY (RADIAN 0.1) while (rforce < "detect-contact"))
     (cond ((contact?) (error?)))
     (CHANGE Y by "y" while (yforce < "detect-contact"))
     (cond ((contact?) (error?)))
     (CHANGE X by "x" while (xforce < "detect-contact"))
     (cond ((contact?)) (t (error?))))
```

Figure 5.12 - DROP-INTO strategy and its expansion into LLAMA. The text in italics indicates the parts generated by the Feedback Planner.

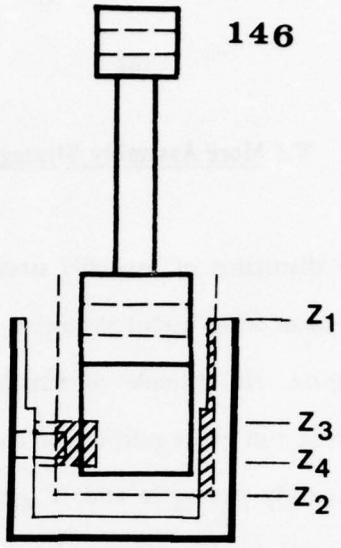


Figure 5.13 - Potential contacts coming down into place for DROP-INTO operation. The range Z1 to Z2 indicates the contacts with the piston. The range Z3 to Z4 indicates contact with the pin. Because the ranges overlap, only the range between Z1 and Z3 is unambiguous.

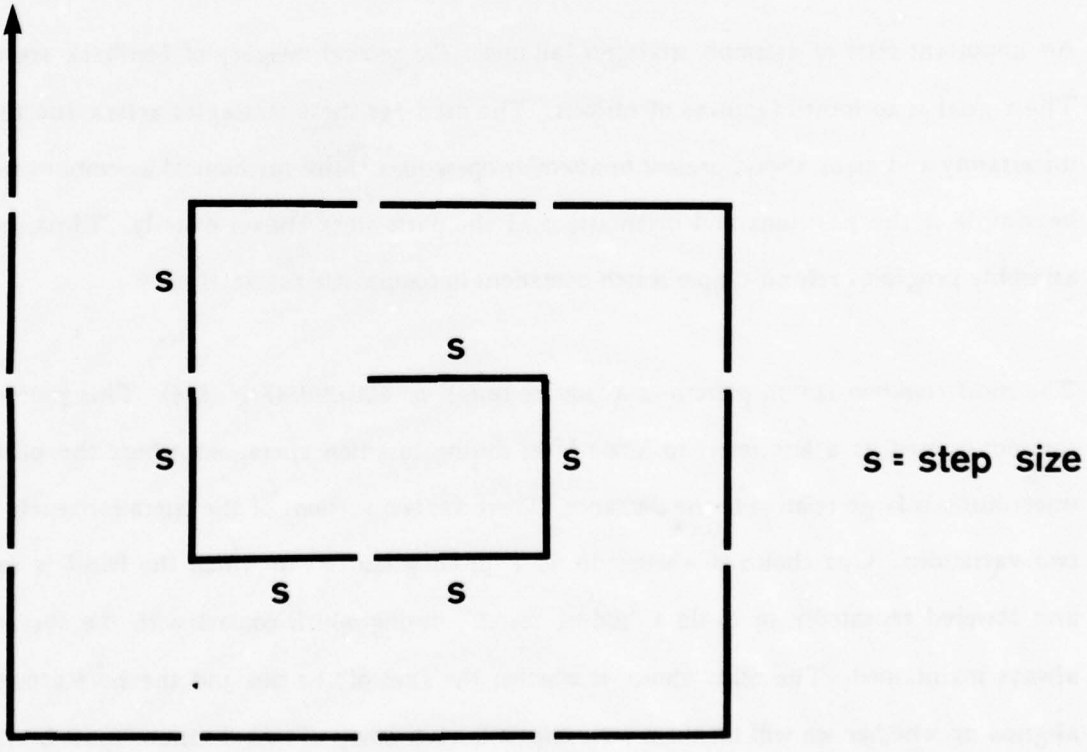


Figure 5.14 - SQUIRAL search pattern, s is the step size.

V.5 More Assembly Strategies

We have, so far, limited our discussion of assembly strategies to the PEG-IN-HOLE strategy. This section briefly discusses some other useful strategies. The strategies themselves have all been tried out in actual programs. No examples of actually instantiating these strategies with the Feedback Planner have been run. The purpose of considering these strategies here is to point out the variety of assembly strategies and to give some feeling for the range of mechanisms needed to instantiate them. We will consider three classes of assembly strategies: (1) feedback searches, (2) grasping strategies and (3) insertion methods.

V.5.1 Feedback Searches

An important class of assembly strategies fall under the general category of feedback searches. Their goal is to locate features of objects. The need for these strategies arises due to the uncertainty and error always present in assembly operations. Most mechanical assemblies would be simple if the positions and orientations of the parts were known exactly. Thus, many assembly programs rely on simple search operations to compensate for variability.

The most common search pattern is a "square spiral" or SQUIRAL (Fig. 5.14). This pattern is commonly used as a last resort to locate holes during insertion operations where the position uncertainty is large relative to the clearance. There are two versions of the operation, each with two variations. One choice is whether to do a "probing search", in which the hand is raised and lowered repeatedly, or to do a "sliding search", during which contact with the surface is always maintained. The other choice is whether the axes of the pin and the hole are to be aligned or whether we will tilt them with respect to each other. These decisions must be made on the basis of (1) the sensitivity of the force feedback, (2) the characteristics of the surface and (3) the clearance between the hole and the pin.

The "probing" type of search makes less demands on accuracy of force sensing than the "sliding" search. In a probe, the hand can push down fairly hard and then detect the large forces generated. In the sliding search, less force must be applied, to avoid catching the tip of the pin on the surface. The manipulator must also be sensitive to small deviations in the contact force in order to insure contact with the surface. The smoothness of the surfaces involved also affects whether the sliding search can be succesful. The clearance between the hole and the pin determines whether the axes should be aligned or tilted. This decision is similar to the decision on "close fit" vs. "loose fit" insertion in [Inoue]. This "loose" vs "tight" fit decision is specified as part of the pre-requisites of the operations. The other descisions must be made by examining the details of the parts and the operation.

The "probing search" operation consists of raising the hand, moving a step and lowering until either contact is made or the hand moves below where the surface should be. If no contact is made, then either we have found the hole or missed the surface altogether. Another motion, downward or sideways relative to the hole, serves to differentiate the two cases. The Feedback Planner has to specify the parameters of the operation and decide what the "test for success" should be. The basic parameter is the size of the steps, which is chosen to be some fraction (about half) of the clearance. The other important parameter is how far the peg is expected to fall when it lands in the hole. This depends on whether the pin and hole are aligned or not. If the clearance is large enough for a "loose fit", then we can expect the pin to go in quite a ways into the hole. Otherwise, the pin can only go in a small distance, which is a function of the angle between the axes. The test for success is to "wiggle" the pin relative to the hole. The resulting motion and forces will indicate if the hole has been found. In the tight fit situation we maintain a downward force while wiggling. This allows us to detect the contact with the sides of the hole.

The "sliding" search involves moving around the surface in a spiral pattern, while maintaining contact. Falling into the hole is detected by the discontinuity, first in the z position and then in the force along the direction of movement, which occurs when encountering the leading and lagging edges of the hole, respectively. It requires specifying the same two parameters as the probing search plus an additional two force parameters, the surface contact force and the "collision" force with the side of the hole. The sliding search can be faster and more reliable than the probing search. There are limits to its applicability. We have mentioned the requirements it makes on good force feedback and surface smoothness. It also requires that there be some surface around the hole to slide on. If the hole is on a thin "ledge", a fairly common situation, then either the search has to be modified to take advantage of the direction of the surface or a probing search must be used.

The decisions that have to be made by the Feedback Planner are:

(1) Which is more appropriate, the probing or the sliding search?

(2) How far is the pin expected to fall into the hole?

(3) What force parameters should be used in the sliding search?

(4) How much "wiggle" should be used in the final test?

The fact that our manipulator, the Silver arm, has sensitive force feedback makes the decision on probing vs. sliding rest solely on the qualities of the surfaces involved. The parts descriptions we have used do not include the information on material needed to deduce smoothness, but this is a minor point. The important information is whether there is enough surface surrounding the hole for a sliding search to happen at all. If the hole is on a ledge, the probing search is preferable. This can be determined by simulating the contact history of an object, whose range of positions is the area of the search, over the surface. If contact with the surface cannot always be maintained, then the probing search can be chosen.

Predicting how far the pin will fall in the hole is done by simulation. We detect the first contact beween the pin and the sides of the hole. The force parameters are standardized for the motions, they are reduced if there is a possibility of the object in the hand slipping. The size of the "wiggle" is chosen to insure contact between the pin and the sides of the hole.

V.5.2 Grasping Strategies

When we considered the pick and place operation in Chapter 4, we assumed that the position and orientation of the object to be grasped was known almost exactly. That is, in fact, the case when we are grasping objects that have been previously placed at their positions by the manipulator. When first grasping objects, the range of uncertainty in position is much greater. The computation of the sets of legal grasp positions is still necessary, but we also need some strategies for locating the desired grasp point.

The most common method for grasping a standing cylinder or a cuboid with dimensions smaller than the maximum opening of the fingers, consists of first grasping it with the hand rotated at 90 degrees of the desired grasp position. Then rotating the hand and grasping again. This centers the object with respect to the hand.

The method for grasping either a cylinder on its side or an elongated cuboid is to grasp them along the smaller dimension and then tap one end against a known position. The tapping reduces the uncertainty along the length. The Feedback Planner must predict where on the object the contact will occur; from this the position of the hand on the object can be computed.

Both of these methods require simulating the grasping operation taking into account the uncertainty in the positions of the objects. We now recognize this as the prime function of the Feedback Planner.

V.5.3 Insertion Methods

The first two phases in the PEG-IN-HOLE strategy align the peg and the hole and insert a very small part of the peg into the hole. There are several options as to what to do next. The simplest is PUSH-INTO, in which two axes of the hand are servoed to zero forces and a small insertion force is applied along the axis of the cylinder. The alternatives are either to rotate the hand so as to tighten a screw to a specified torque; and for cases with very tight fits and/or friction, WIGGLE-INTO might be required. All these are very simple strategies that perform a simple motion until a force or torque threshold is exceeded. The only parameters that need to be specified are the depth of insertion and the forces involved.

V.6 User Interactions

We have, so far, almost totally ignored the users' role in the system. This section develops the role of the user in a little more detail.

V.6.1 Levels of Performance

The ultimate assembly system would take a sample of the object to be assembled and from it, would generate a program to assemble it under factory conditions. Needless to say, this is currently not feasible; nor is it likely to be for a good many years. What we want is to isolate the different components of this Utopia system and rank them in importance. We can then design systems that provide the most useful performance that is technologically feasible. We will start by listing them in decreasing order of "utopianess".

The ability to acquire object decriptions directly from the objects is probably the hardest problem in the Utopia system. This requires, besides sophisticated vision, the ability to carry

out the necessary measurements. On the other hand, this ability is probably superfluous. Objects still have to be designed by humans, this involves engineering specifications; either to a design automation system or via draftsman. The design automation system is likely to employ an internal description that is either directly suitable or readily convertable to a representation useful to an automatic assembly system. Moreover, research on interpretation of engineering drawings [PADL] is already in progress.

The next most difficult part is determining how an object should be assembled, merely from a description of its parts. This is a very difficult problem, but a start has been made on this goal. It takes the form of two programs being developed at the MIT AI Laboratory. One [Galkowski], examines the descripion of the parts and generates a series of constraints on how the parts could go together. The other [Freiling], embodies knowledge on how simple machines work. This could eventually develop into a mechanism to further constrain the assembly by using expected function information.

We have by-passed both of these problems in the design of LAMA and have assumed that the parts are described to the system directly and that assembly instructions are available. The level at which these instructions can be usefully employed depends on the development of three types of capabilities. A natural language interface suitable to assimilating the instructions, an understanding of the physical interactions of parts and a planning system capable of exploiting this type of knowledge. All of these seem feasible within a period of five years.

Then comes the level of expertise embodied by the prototype LAMA design, without the Assembly Planner. A working prototype probably requires on the order of one or two man-years of work.

V.6.2 The Role of the User

As we move down the list of systems mentioned above, the participation of the user in the development of the manipulator programs increases. In the Utopia system, the user needs to know how to start the system and nothing else. A little below this, all that is involved is describing the assembly.

The full LAMA design calls for the user to be able to describe the objects using the system's vocabulary of basic objects and relations and to describe the assembly to a limited natural language interface. This requires only moderate amounts of skill. On the other hand, it also requires a user to specify the assembly strategies to be employed. The most common ones will be available to the system from the start, but these are likely not to be completely adequate. This is a task for an expert; but probably many such people are not required. This could be considered a part of the system's maintenace.

The more limited LAMA design makes extensive appeal to the user's expertise. The goal of the design is to make that expertise not be in programming. The user is required to cooperate with the system by specifying the assembly plan and then answering the system's questions concerning the likelihood of motions and finally to make decisions on error correction. The system detects the possibilities of error, it is up to the user to decide what to do. I have not addressed the interfacing problem at all.

The system's array of assembly strategies will not sufficient for all tasks. This can be remedied by either defining more strategies for general use or merely writing programs in the strategy language. Both of these tasks require expertise in assembly operations as well as in the use of the system. Even at the level of the strategy language, the system can make a significant contribution towards the final program.

V.7 Summary

We have described the mechanisms used by LAMA to integrate feedback techniques into an assembly operation. The basic unit of knowledge for assembly techniques is the *assembly strategy* as embodied in the *skeleton programs*. These programs are then expanded by the system to take into account the particular geometric environment. The basic tool used during the expansion process is *qualitative simulation* where the interactions of uncertainty in position, forces, contacts and motions are roughly determined. This process provides the information necessary to evaluate the feasibility of a proposed assembly operation as well as simplify the interpretation of the feedback information obtained during execution. The implementation of the ideas discussed in this chapter is still in the early stages. A crude implementation of the contact history computation and the code generation phase exists but it is very limited in generality and it is still unreliable. Much more work needs to be done to make it a usable system.

VI. The Assembly Planner

The assembly descriptions we saw in Chapter I require much processing before they can become manipulator programs. The first step in that process is to transform the initial assembly description into a complete **assembly plan**. The assembly plan is a sequence of steps taken from the inventory of "primitive" assembly steps e.g. grasping, peg-in-hole insertion, place-in-vise, etc. The initial description can be seen as a sparse sub-sequence of the assembly plan in which some of the steps are not fully specified. The Assembly Planner tries to construct a fully specified sequence of assembly steps that will contain the initial description and fulfill the pre-requisites of the individual assembly steps.

VI.1 The Scope of the Problem

The problem of construction planning has long held the center stage in the study of **problem solving**. During the last three or four years several programs have been developed to do construction planning in the Blocks World domain. Fahlman's BUILD [Fahlman] is expert in planning Blocks World assemblies. The programs of Sussman and Sacerdoti although not as expert as BUILD have explored general issues of planning and debugging in the context of assembly problems. [Sussman PhD] treated assemblies of blocks exclusively while [Sacerdoti], as a part of SRI's Computer Based Consultant project [Nilsson], has also considered the assembly (and disassembly) of a water pump.

Of these programs, only BUILD considered the issues of stability, contact, etc. which are vital to the process of mechanical assembly. But even BUILD, being limited to blocks structures could ignore most of the problems of spatial interactions. Sacerdoti's use of NOAH in the SRI's

Consultant project avoids all these problems because it assumes a human as the manipulator. These programs, on the othe hand, have examined many of general planning problems such as control, debugging and interactions. I will therefore avoid most of these general issues by restricting the nature of the problem and focus instead on the problems of geometry and physics that have been ignored elsewhere.

In the full blown assembly planning problem we are given the initial and final state of the parts and we are asked to generate a sequence of primitive operations which will transform one into the other. I will not consider the problem at this level of generality. Instead, consider the solution (call it the **assembly plan**) to the full problem mentioned above. It consists of a sequence of operators with specified arguments each of which is applicable to the state produced by the preceding operation. We will assume that the input to the assembly planner is a sub-sequence of this full operator sequence. Contiguous operators in the sub-sequence need not be contiguous in the full sequence. Furthermore, the operator applications need not be fully described. The parts they apply to must be specified but the orientations and positions may not be. A further restriction is that any operator in the full sequence that establishes a relationship between two parts must be included in the sub-sequence. This last restriction implies a "complete" description of the assembly in terms of the parts. Notice that it does not require any manipulator operations such as grasping and clamping to be specified.

These constraints limit the scope of the problem enormously. Many might think that it makes this planning problem uninteresting. At the level at which assembly planning has been normally done this is quite correct and, of course, it is precisely the point of the restrictions. On the other hand, it still leaves many problems which have to be solved before a complete manipulator system can be written. It is on these problems I want to focus. Notice that the sample assembly description we considered briefly in Chapter I satisfies the restrictions stated above. I believe the problems remaining to be non-trivial. I also believe that the assembly

descriptions as constrained above correspond to the level of detail that instructions are normally given to humans doing assemblies.

VI.2 A Scenario for the Assembly Planner

In this section we will consider how the assembly description we saw in chapter 1 (repeated below) can be transformed into an assembly plan. Later we will briefly discuss the techniques employed in the scenario. I must emphasize that the following scenario is purely speculative. It is included here for the light it sheds on the design of the system.

The initial assembly description we will consider is the following:

(1) Insert the piston pin partway into the piston.

(2) Place the rod's small end on the piston pin inside the piston.

(3) Push the pin through the rod and the piston hole.

This assembly description must be expanded into a sequence of operators taken from the following list: GRASP, UNGRASP, PLACE-IN-VISE, INSERT and PUSH-THROUGH. Notice that each sentence in the initial description specifies the application of one operator and that the physical relationships between the parts in the final assemblies are all explicitely achieved. We must now expand this description to the point where it specifies which objects to grasp and which to clamp, what the orientation of the vise should be, etc. This level of detail is required to fully specify the operators. We will still ignore manipulator motions necessary to cope with error in the manipulator and uncertainty in the positions and orientations of the parts. We will assume that the assembly steps can be reliably executed once their pre-requisites are met.

The major problem we must face is that of uniquely specifying the arguments to the operators. The first assembly step calls for inserting the pin partway into the piston. It does not specify which part is to be held in the hand or what is to be used to hold the other part. Neither does

it restrict the orientation of the parts with respect to the manipulator; yet the next step requires knowing all these facts.

One of the prerequisites for the insertion operation is that one object be in the hand and the other be clamped on the vise or a jig in such a way that it will not move under the forces generated during insertion. We can choose to place the piston in the vise and insert the pin into it. This choice makes it impossible to insert the piston rod onto the pin since nothing is holding the pin. Thus we must place the pin into the vise and insert the piston onto it there.

The next operation, pushing the pin through the rod and the piston can be performed in several ways. We can hold the piston or the piston rod in the hand and move the pin either by pushing it against something solid or by clamping it in the vise. Alternatively we can hold the rod or the piston in the vise and push the pin with the hand. The first thing to notice is that all these methods require that the pin be free to move; so we must first remove it from the vise. We again have the choice of whether to grasp the piston rod or the piston. The planning system has to recognize that the connection between the piston rod and the rest of the subassembly is not very sturdy. This implies that the assembly should be grasped by the piston. Having decided this we can then reduce the number of choices of how to carry out the insertion of the pin. We now have only to choose between pushing the pin against a solid object or placing it in the vise. Both are equally suitable so no decision is made at the moment.

We have ignored many important details so far. When placing the pin in the vise we have to specify its orientation and displacement relative to the vise. We also have to decide on the orientation of the vise. For the moment we will assume that the vise can only be positioned at -90, 0 or +90 degrees from vertical. We can grasp the piston one of three ways (Fig 6.1). If we know its original orientation we can decide how to grasp it and also how to orient the vise.

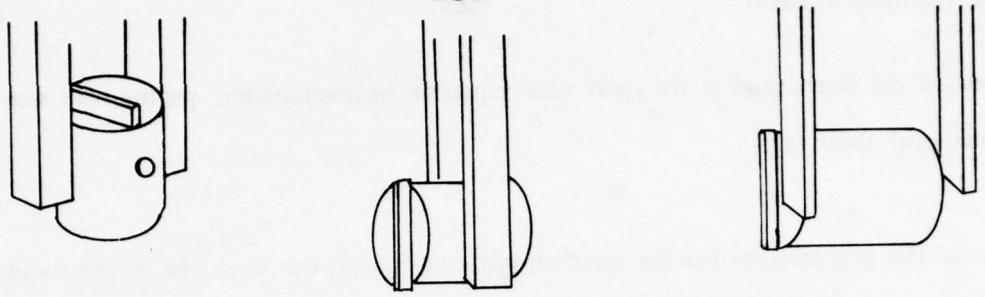
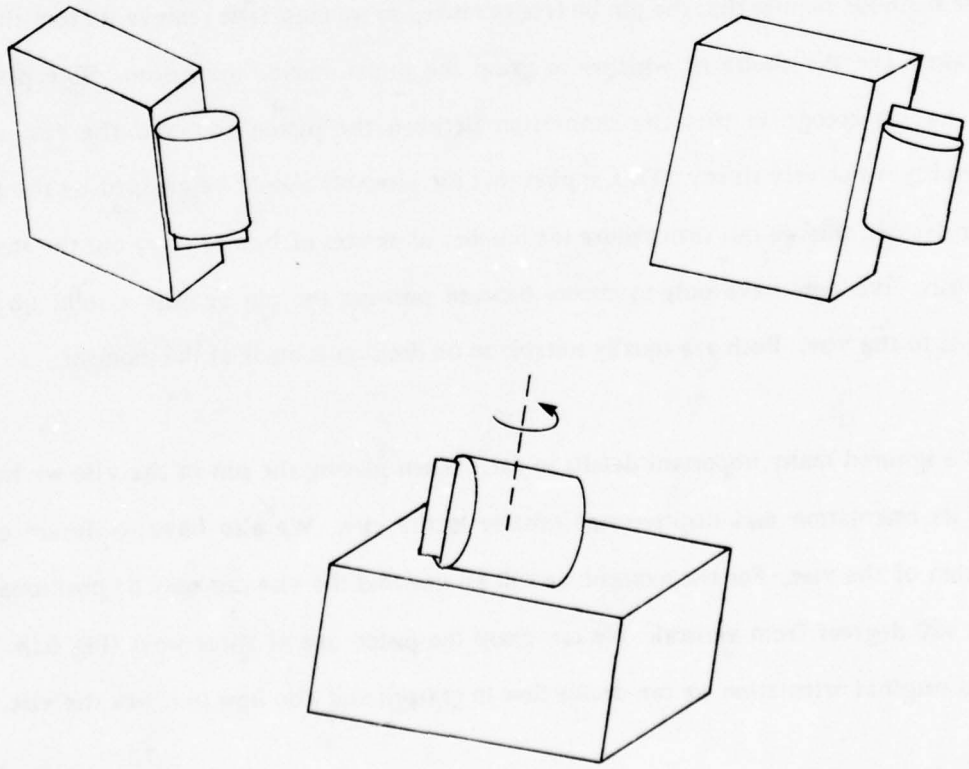


Figure 6.1 - Three ways of grasping the piston.

Figure 6.2 - Stable orientations of piston on piston-pin.



Once the piston is on the pin, it inherits the pin's degrees of freedom plus its own freedom to rotate on the pin. If we consider the orientation of the piston on the pin to be that uncertain we cannot place the piston rod onto the pin. We can ask the user what the stable orientations of the piston on the pin are at the different vise orientations. The stable piston orientations are indicated in Fig. 6.2.

The focus now shifts to the insertion of the rod onto the pin. The possible orientations of the piston constrain the orientations of the rod. Orientations of the piston in which the cavity is facing downward cannot be used since they do not allow a path for the rod. Our choice is then whether to keep the vise vertical or horizontal. We can examine the requirements on the next operation to see if we can make the decision. We have already seen that we want to grasp the piston during the process of pushing the pin through the rod and the piston. If the vise is vertical when we take the assembly out of the vise after inserting the rod onto the pin then the pin would be unsupported. Preventing this would require the vise to be horizontal; but we could have performed the insertion with the vise horizontal in the first place. Then we could just take the pin out of the vise and push it through the piston.

Fig 6.3 shows the complete assembly plan to be produced. It is also the kind of description that could be input to the system directly.

VI.3 Assembly Planning as Constraint Satisfaction

The scenario above has tried to emphasize a view of assembly planning as a process of satisfying constraints. Each operation imposes some constraints on the assembly process. It also has several degrees of freedom which can be decided upon by examining the constraints imposed by other operations. This whole process can be carried out at several levels of increasing detail. In the scenario above I indicated two levels of detail. We first considered

```
(GRASP OBJ: [PISTON-PIN])

(PLACE-IN-VISE OBJ: [PISTON-PIN])

(UNGRASP OBJ: [PISTON-PIN])

(GRASP OBJ: [PISTON]
       SUCH-THAT (FACING+ ([PISTON] TOP) DOWN))

(INSERT OBJ1: [PISTON-PIN]
        OBJ2: [PISTON PIN-HOLE]
        SUCH-THAT: (PARTLY (FITS-IN OBJ1 OBJ2)))

(UNGRASP OBJ: [PISTON])

(GRASP OBJ: [ROD]
       SUCH-THAT: (FACING+ ([ROD-BAR] TOP) UP))

(INSERT OBJ1: [PISTON-PIN]
        OBJ2: [ROD SMALL-END-HOLE])

(UNGRASP OBJ: [ROD])

(GRASP OBJ: [PISTON])

(REMOVE-FROM-VISE OBJ: [PISTON])

(PUSH-INTO OBJ: [PISTON-PIN]
           SUCH-THAT: (AND (FITS-IN [PISTON-PIN] [PISTON PIN-HOLE])
                           (FITS-IN [PISTON-PIN] [ROD SMALL-END])))

(UNGRASP OBJ: [PISTON])
```

Figure 6.3 - Assembly Plan for the piston assembly

questions of which objects were going to be in the vise and which in the hand. Then we considered orientations of parts and detailed choice of grasp point.

This approach relies on answers to the following types of questions. The last two chapters presented, in different contexts, techniques which are adequate for answering the first six types of questions.

(1) What are the options for performing each operation? This amounts to different strategies for the operation. This is part of the description of the operation.

(2) What are the constraints on the motion of the parts involved in an operation? This, again, belongs in the description of the operation.

(3) What are the degrees of freedom of a part? This, in principle, can be determined by examining the relationships that have been established between parts by the different operations. In practice, parts that are not directly related can also constrain each other. Simulating small motions can give a fairly good idea what the constraints on the object are.

(4) What are the interactions between degrees of freedom of different objects? For example, realizing how the orientation of the piston on the pin constrains the approach of the piston rod.

(5) What are the options on grasping and clamping parts? At this level, we are only interested about hwo these decisions affect the feasibility of the operations. This depends mostly on the orientation of the parts rather than on exactly where they are grasped. The grasping computation at this level is limited to which grasp sets are accessible at all. This reduces the complexity of the problem.

(6) Are there any collision free trajectories to a specified location given the ranges in the position of the neighboing parts? The traditionally difficult problem of finding collision free trajectories becomes tricky when the positions and

orientations of the parts are not fixed. This requires the ability to divide the situations into cases where the techniques of Chapters 4 and 5 are again applicable. (7) Which configurations of parts are stable? This is a very difficult problem. Fahlman implemented a numerical formulation in the Blocks World domain. I believe that Fahlman's numerical approach is not feasible with more complicated parts. A mixture of numerical, heuristic and experimental approach seems to be called for.

VI.4 Conclusions

This chapter has investigated some of the mechanisms that might be used to convert an incomplete assembly description into an assembly plan. This process is extremely important from a theoretical standpoint. It is also highly desirable from a strictly practical viewpoint. Being able to describe an automatic mechanical assembly at a level comparable to a description of the assembly for a human would go a long way towards making automatic assembly systems feasible. On the other hand, an examination of what is required for implementing the scenario in this chapter makes it clear how difficult even this limited form of the goal is. One of the most difficult requirements to meet is the ability to predict stability of complex assemblies. The approach we hinted at earlier, i.e. consulting the user on such problems is misguided. It might be more convenient for a human to provide a complete assembly plan than to sit there and answer stability questions.

Our approach for the short range is to assume that the assembly plan is available and isolate the assembly planning problem from the rest of the system. The goal of this chapter has been to define the problem well enough so that independent research can proceed

VII Concluding Remarks

This chapter provides a brief summary of the main ideas in this report and a list of topics suitable for further reasearch.

VII.1 Summary

The preceeding chapters have described a design and partial implementation of a system, LAMA, whose goal is to simplify the process of specifying an automatic mechanical assembly. The system provides mechanisms for describing the objects to be assembled and the procedure to be used, at a level which is convenient for the user. It is still the responsability of the user to plan the assembly; but the system helps write the manipulator program.

The system's knowledge of assembly operations is embodied in a set of skeleton programs which indicate the feedback strategy to use in achieving a particular assembly state. These programs are supplemented by a description of the desired assembly state produces by each manipulator motion. This allows the system to specify the parameters in the program and include error checking.

LAMA's expertise is spatial. It relies on qualitative spatial simulation to discover contraints on motions and positions. These constraints are then exploited for many of the system's operations. The LAMA design has tried to be as sparing in the number of mechanisms as possible. Most of the burden of the system has been placed upon the spatial modeling routines. I believe this to be a good design choice since our goal is to convert a nebulous problem into a well defined one that has solutions.

VII.2 Problems for Further Research

I believe that this research has been succesful in formulating many of the problems involved in a mechanical Assembly environment. It has been less succesful in solving them. Hopefully, that is a temporary state of affairs. This section will attempt to summarize the problems I have tried to illustrate in the preceeding chapters. First I will present the major problems, the ones that limit the whole structure of the system. Then, I will discuss the technical problems that can probably be solved within the structure of the current system design.

These are the major design problems:

(1) First is the spatial modeling operations. The space filling list structure representation has served me fairly well, but many problems remain. A much better way of approximating objects by aligned rectangles is needed. This is the bottleneck of the whole system as it stands now. If the method cannot be improved the reliance on the space filling approach should be reversed. Hopefully, the rest of the design does not depend on the manner of doing the spatial operations of union, instersection and difference.

(2) The next major problem is the physical modeling. I have adopted a very weak physical model, not capable of real prediction. This is partly a philosophical objection to the use of numerical models and partly a reaction to the lack of convenient ones. Whatever techniques are necessary to achieve adequate physical models should be employed. I believe this will eventually involve active, machine supervised experimentation.

(3) The other major problem is the planning stage. Chapter III outlines the kind of operations needed to carry out the limited planning I believe to be convenient for a Mechanical Assembly System. That outline needs a lot of work before it becomes a viable planning system.

(4) Then there is the use of visual feedback. This is a highly desirable addition to many of the operations of an assembly system. The effective use of vision in any environment requires large amounts of work but in the end I believe it will prove to be very rewarding.

There are many technical problems which have not been settled throughout this report. I will try to list them here in groups. First the modeling system:

(1) Experiments should be performed to identify the kinds of constraints that people find convenient to use in describing objects and assemblies.

(2) A more general implementation of the constraint system using more primitive objects.

(3) A more compact representation for objects is needed.

I will not say anything about the planning system since all of that will probably require reformulation. The pick and place operation still has many problems:

(1) LGSETs need generalizing to include the missing rotational degree of freedom. PGSETs also need generalizing to include the extra translation.

(2) Consideration of movable obstacles during grasping. Some obstacles detected during the grasping operation can be moved, if necessary for grasping, e.g. the piston rod when grasping the rod, pin, piston subassembly in the vise.

(3) The grasping computation totally ignores holes. This is safe only as long as the holes do not remove part of the surface to be grasped. Treating holes as constraining objects might be an adequate solution.

(4) A more general model of the manipulator needs to be considered. The "disembodied hand" type of manipulator is not completely general.

(5) A better approach is needed to the interaction between paths to objects and the legal grasp positions. Our solution of using a standard approach from the top,

while adequate, leaves much to be desired.

(6) A better collision avoidance strategy is needed which considers more than one direction in avoiding strategies and considers the motion constraints on the objects.

Most of the problems remaining for the Feedback Planner are the major ones of better spatial and physical modeling as well as the use of vision. Some local problems can be identified also:

(1) Determining a grasp strategy that makes use of the knowledge of expected obstacles to identify the position and orientation of the object.

(2) More focus on error correction rather than error detection could be achieved by more thorough use of the comments on manipulator motions.

(3) The treatment of merging conditionals must be made more complete. Some ways to deal with the combinatorial aspects of conditionals should be devised.

(4) Loops must be considered. I believe an extension of the methods used for modeling uncertain positions will be fruitful for some types of loops. This involves simulating all the loop operation in parallel by treating the object as if it were simultaneously in all the positions generated by the loop. Other loops will require simulating a typical operation rather than all the operations.

(5) Multiple axis motions have to be handled. The effects of multiple dependent motions can be complex because the ranges of motions are interrelated.

(6) Knowing that a motion succeds or fails provides additional constraints on the position of objects. The current design ignores this completely.

(7) The ability to consider alternative initial ranges of positions for motions should be explored.

(8) Two handed coordination will provide a fertile ground for more research.

(9) More manipulator independence is desirable throughout.

This wide range of problems indicates a very lively research area.

Bibliography

[Ambler & Popplestone]
A. P. Ambler and R. J. Popplestone, *Inferring the Positions of Bodies from Specified Spatial Relationships*, AISB Summer Conference, University of Sussex, July 1974.

[Ambler et. al.]
A. P. Ambler, et. al., "A Versatile System for Computer Controlled Assembly", *Artificial Intelligence*, Volume 6, Number 2. 1975.

[Appel]
A. Appel, "Modeling in Three Dimensions", *IBM Systems Journal*, Volume Seven, Numbers 3 and 4, 1968.

[Baumgart]
B. G. Baumgart, *GEOMED - A Geometric Editor*, Stanford Artificial Intelligence Laboratory Memo AIM-249, May 1974.

[Binford, et. al.]
T. O. Binford, et. al., *Exploratory Study of Computer Integrated Assembly Systems*, Stanford Artificial Intelligence Laboratory, NSF Report covering November 1975 to June 1976.

[Boberg]
R. W. Boberg, *Generating Line Drawings from Abstract Scene Descriptions*, Unpublished S. M. Thesis, MIT Dept. of Electrical Enginnering, December 1972.

[Bolles & Paul]
R. C. Bolles and R. Paul, *The Use of Sensory Feedback in a Programmable Assembly System*, Stanford Artificial Intelligence Laboratory Memo AIM-220, October 1973.

[Bolles]
R. C. Bolles, *Verification Vision Within a Programmable Assembly System: an Introductory Discussion*, Stanford Artificial Intelligence Laboratory Memo AIM-275, December 1975.

[Braid]
I. C. Braid, "The Synthesis of Solids Bounded by Many Faces", *Communications of the ACM*, Volume 18, Number 4, April 1975.

[Darringer & Blasgen]
J. A. Darringer and M. W. Blasgen, *MAPLE: A High Level Language for Research in Mechanical Assembly*, IBM Research Report RC-5606, September 1975.

[Donelson]
Bill Donelson, "Efficiency Coding for Objects in a Large 3-D Data Base", *Architecture Machinations*, Volume II, Numbers 15, 16, and 17, MIT Department of Architecture, Architecture Machine Group, April 1976.

[Eastman]
C. M. Eastman, "Representations for Space Planning", *Communications of the ACM*, Volume 13, Number 4, April 1970.

[Fahlman]
S. E. Fahlman, *A Planning System for Robot Construction Tasks*, MIT Artificial Intelligence Laboratory Technical Report 283, May 1973.

[Finkel, et. al.]
R. Finkel, R. Taylor, R. Bolles, R. Paul and J. Feldman, *AL, A Programming System for Automation*, Stanford Artificial Intelligence Laboratory Memo AIM-177, November 1974.

[Freeman & Shapira]
H. Freeman and R. Shapira, "Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve", *Communications of the ACM*, Volume 18, Number 17, July 1975.

[Freiling]
M. Freiling, *The use of a hierarchical representation in the understanding of mechanical systems*, forthcoming PhD Dissertation, 1977.

[Galkowski]
J. T. Galkowski, *Deriving Assembly Constraints from Parts Specifications*, Unpublished S.M. Thesis, MIT Department of Electrical Engineering, June 1976.

[Grossman]
D. D. Grossman, *Procedural Representation of Three Dimensional Objects*, IBM Research Report RC-5314, March 1975.

[Grossman & Taylor]
D. D. Grossman and R. H. Taylor, *Interactive Generation of Object Models With a Manipulator*, Stanford Artificial Intelligence Laboratory Memo AIM-274, December 1975.

[Horn & Inoue]
B. K. P. Horn and H. Inoue, *Kinematics of the MIT-AI-VICARM Manipulator*, MIT Artificial Intelligence Laboratory Working Paper 69, May 1974.

[Hosaka et. al.]
M. Hosaka, F. Kimura and N. Kakishita, "A Unified Method for Processing Polyhedra", *IFIP Congress*, 1974.

[Inoue]
H. Inoue, *Force Feedback in Precise Assembly Tasks*, MIT Artificial Intelligence Laboratory Memo 308, August 1974.

[Lavin & Lieberman]
M. A. Lavin and L. I. Lieberman, *A System for Modeling Three-Dimensional Objects*, IBM Research Report RC-5765, December 1975.

[Lieberman & Wesley]
L. I. Lieberman and M. A. Wesley, *AUTOPASS, A Very High Level Programming Language for Mechanical Assembler System*, IBM Research Report RC-5599, August 1975.

[Moran]
T. Moran, *Structuring Three Dimensional Space for Computer Manipulation*, Dept. of Computer Science working paper, Carnegie-Mellon Univ., June 1968.

[Nevins et. al.]
J. L. Nevins, et. al., *Exploratory Research in Industrial Modular Assembly*, Charles Stark Draper Laboratory, NSF Report covering December 1974 to August 1975.

[Nilsson]
N. J. Nilsson, ed., *Artificial Intelligence - Research and Applications*, Stanford Research Institute, ARPA Progress Report, May 1975.

[Nilsson 1969]
N. J. Nilsson, "A Mobile Automaton: An Application of Artificial Intelligence Techniques", *Proceedings International Joint Conference on Artificial Intelligence*, May 1969.

[PADL]
An Introduction to PADL, Production Automation Technical Memorandum 22, University of Rochester, December 1975.

[Pfefferkorn]
C. E. Pfefferkorn, "A Heuristic Problem Solving Design System for Equipment or Furniture Layouts", *Communication of the ACM*, Volume 18, Number 5, May 1975.

[Pfister]
G. F. Pfister, *On Solving the FINDSPACE Problem, or How to Find Where Things Aren't ...*, MIT Artificial Intelligence Laboratory Working Paper 113, March 1973.

[Popplestone]
R. J. Popplestone, *How Could FREDDY Put Things Together*, Dept. of Machine Intelligence and Perception, University of Edinburgh, Memo MIP-R-88, May 1971.

[Rich and Shrobe]
C. Rich and H. Shrobe, *A LISP Programmer's Apprentice*, MIT Artificial Intelligence Laboratory Technical Report, December 1976.

[Roberts]
L. G. Roberts, *Homogeneous Matrix Representation and Manipulation of N-Dimensional Constructs*, Document MS1045, Lincoln Laboratory, MIT, May 1965.

[Rosen, et. al.]
C. Rosen, et. al., *Exploratory Research in Advanced Automation*, Stanford Research Institute, NSF Report, January 1976.

[Sacerdoti]
E. D. Sacerdoti, *A Structure for Plan and Behavior*, Stanford Research Institute Artificial Intelligence Center Technical Note 109, August 1975.

[Silver]
D. Silver, *The Little Robot System*, MIT Artificial Intelligence Laboratory Memo 273, January 1973.

[Sussman]
G. J. Sussman, *The FINDSPACE Problem*, MIT Artificial Intelligence Laboratory Vision Flash 18, August 1971.

[Sussman PhD]
G. J. Sussman, *A Computer Model of Skill Acquisition*, MIT Artificial Intelligence Laboratory Technical Report 297, August 1973.

[Taylor]
R. H. Taylor, *A Synthesis of Manipulator Control Programs From Task-Level Specifications*, Stanford Artificial Intelligence Laboratory Memo AIM-282, July 1976.

[Waters]
R. Waters, *A Mechanical Arm Control System*, MIT Artificial Intelligence Laboratory Memo 301, January 1974.

[Will]
P. M. Will, *Computer Controlled Mechanical Assembly*, IBM Research Report RC-5428, May 1975.

[Will & Grossman]
P. M. Will and D. D. Grossman, "An Experimental System for Computer Controlled Mechanical Assembly", *IEEE Transactions on Computers*, Volume C-24, Number 9, September 1975.

[Winograd]
T. Winograd, *Procedures as a Represeniation for Data in a Computer Program for Understanding Natural Language*, MIT Artificial Intelligence Laboratory Technical Report 17, February 1971.

[Winston]
P.H. Winston, *Proposal to the Advanced Research Project Agency*, MIT Artificial Intelligence Laboratory Memo, May 1976.

App 1. LLAMA: The Target Language for the LAMA system

This appendix specifies **LLAMA** (Low-level Language for Automatic Mechanical Assembly). This language is designed for use with the Little Robot System [Silver]. The language definition itself is independent of the manipulator, but the language was defined with one manipulator in mind. A more general manipulator will undoubtedly require extensions to the language. Similar languages for more general manipulators are described in [Finkel et al] and [Darringer & Blasgen].

The **LLAMA** "language" merely extends LISP with a few primitives for arm control. The new statements are of three types which I will call **reference frame**, **single axis motion** and **multiple axis motion** statements.

Manipulator languages are mainly concerned with positions and orientations. The most common and effective method for specifying positions and orientations is by means of reference frame transformations. A **reference frame**, (**reff**), is defined by specifying three rotations and three scalar displacements and a base coordinate system (cf. Chapter 2). There are two global base coordinate systems known as **ARM-REFF** and **TABLE-REFF**, the first is the manipulator's coordinate system centered between the fingers, the other is anchored on the table and all reffs are related to it, usually through several levels of indirection.

The language provides a mechanism for specifying reffs and for linking them together. These are the **REFF** and **ATTACH** statements. The **REFF** command takes a list of angles, a list of displacements and a reff and defines another reff. The **ATTACH** statement indicates that two reffs are to maintain the same relationship to each other, independent of what happens to either

of them. This is most useful for specifying positions and orientation on movable objects. Some examples:

```
(SETQ F1 (REFF (INCH (0.5 0.0 0.5)) (DEGREE (0. 0. 90.)) TABLE))
```

This specifies a reff as the value of F1 which is the table's reff rotated 90 degrees around the z axis and displaced half an inch in both the x and z directions. We can now connect this reff to the manipulator and have it be changed as the manipulator moves by merely doing:

```
(ATTACH F1 ARM)
```

All positions and orientations are specified relative to a reff, which is the value of the variable LOCAL-REFF, defined either in the procedure definition or through the argument list. In our system, the LOCAL-REFF can only be composed by translations and rotations by angles multiples of $\pi/2$ radians. The parameters for the hand, x, y, z, rx, ry, rz (r indicates rotation), and g (the grip distance). There is also a vise with parameters, vrx, vry, vrz (vr for vise rotation) and vg (for the vise grip).

The Little Robot hardware has limited rotational degrees of freedom. Only rotations of the hand about its z axis and of the vise around the table's y axis are possible. Any motion command that calls for a rotation relative to a LOCAL-REFF that corresponds to an unavailable degree of freedom generates an error.

The single axis motions affect only one of the position or orientation parameters of the hand or vise in the LOCAL-REFF. Multiple axis motions affect several parameters. Multiple axis motions are simply several single axis statements executed simultaneously.

The motion statements are of two types: relative and absolute motions. The relative motion statements begin with the keyword CHANGE. Absolute motions are specified by MOVE. The legal single axis motion statements are:

(CHANGE *axis* BY *amount* [{WHILE, WITH} *force-option*])

(MOVE *axis* TO *position* [{WHILE, WITH} *force-option*])

The square brackets [ ] indicate optionally specified arguments. Curly brackets { } indicate disjunctive choice sets. The *force-option* is a boolean combination of force tests. Each force test specifies a condition { <, >, = } on a force parameter {xforce, yforce, zforce, rforce} and a threshold (an arbitrary LISP expression).

The keywords WHILE and WITH indicate the two interpretations of the force option. WHILE specifies that the motion is to continue only as long as the force tests are true. WITH indicates that the arm should move in such a way as to make the force condition true. In that case, the position specification is optional and serves as a maximum rather than a desired value. These type of motions are called **force-driven**.

Multiple independent motions are specified as follows:

(SIMULT *single-axis-motion single-axis-motion* ...)

The SIMULT indicator specifies that each of the motions, specified as above, should be done simultaneously.

Aside from these specific extensions, any LISP statement is also a legal LLAMA statement. The LISP operations can have other primitive statements embedded, e.g. conditional statements can decide between alternative motions.

The only side-effect of the motion statements is to change the position of the manipulator. The axes not specified in a motion are kept at their values before the statement was executed.

Figure 5.12 in the text shows an example of a LLAMA program.

Appendix 2: Spatial Modeling

Programs which must manipulate physical objects perform a variety of basic operations on the descriptions of these objects. A very important subset of these operations consists of the operations INTERSECTION [Lieberman & Wesley] [Hosaka et. al.], and FINDSPACE [Winograd], [Sussman], [Fahlman], [Pfister]. The former consists of determining the intersection volume of two arbitrary solids while the latter involves finding where to put an object on a cluttered table. The first phase of the FINDSPACE operation, the proposer, must examine the current state of the table and suggest a location for the object. The second phase, the verifier must determine if there are any conflicts at that location. This second phase is an intersection operation. The intersection operation is basic to the process of finding collision free trajectories in manipulation systems. Systems for architectural planning [Pfefferkorn] employ both of these operations.

The traditional form for object representations is derived from the representations of objects used in computer graphics. This involves representing the lines, verteces, and sometimes planes, of the objects. This representation is very compact but it has several important conceptual problems. These problems include the failure to represent empty space directly and to store adjacency information. These deficiencies make the design FINDSPACE proposer, which looks for empty space, and INTERSECTION, which tries to determine overlapping volumes, difficult.

This appendix considers an alternative approach to representing objects, the space filling approach. The approach is to divide space into small cells and to associate either the name of an object or empty space with each cell. Empty space gets treated similarly to filled space and

the adjacency of cells is explicitely represented by ordering, as in an array.

This appendix presents particular data structure for space filling representations, based on work reported by Eastman [Eastman]. This representation inherits the basic deficiencies of space filling approaches. The major one is the fragmentation of objects not aligned with the coordinate axes. A related problem is the necessity of using a fairly expensive algorithm to determine an approximation for objects in arbitrary orientation. This makes the insertion and deletion operation for this representation slower. Graphics type representation make the insertion and deletion very simple but the intersection and FINDSPACE operation complex. I believe the balance weighs in favor of the space filling approach.

A2.1. Space Filling Representations

The most primitive version of a space filling representation is an array. Each entry in the array contains a number corresponding to the object which occupies that unit block of space. We will let a zero indicate empty space. The desired resolution of the representation determines the actual volume that a unit block represents.

We will see that this simple scheme proves inadequate for all except the most trivial applications. Before pursuing the inadequacy of this simple representation let's consider some terminology which identifies the components of any such scheme. This will enable us to talk about some generalizations of the naive scheme.

We will refer to the basic unit of storage of a space filling representation as a domain. We have, heretofore, called this a "block of space". Another useful distinction is between an internal domain and an external domain. The former refers to the systems representation of the latter. The shape of a domain is, in principle, arbitrary though we will restrict ourselves to

aligned rectangular domains. These domains will be specified by two coordinate points. The first of these is the origin and the other is the end. The origin is a list of the minimum values of the coordinates of the external domain. The end is a list of the maximim values. We will need and addressing function to map from these external coordinates to the address of the internal domain containing them. The format of the internal address will depend on the implementation. We will refer to it as the domain pointer. We will also need an adjacency function which operates on a domain pointer and returns the neighboring domain's domain pointer. The addressing function need not be one-to-one or even time independent.

For the simple array implementation described earlier the external domains are fixed size cubes. The internal domains are single entries in the array. The domain pointers are 3-tuples of integers which serve as indeces into the array. The adjancency function involves incrementing one or more of the indeces in the domain pointer. The addressing function returns the quotient of the coordinates by the fixed size of the primitive external domain.

The major problem with this simple representation is that it places great burdens on both storage and computation. The domains are fixed in size so any increase in accuracy entails increasing the total number of domains stored. Since operations are performed on a domain by domain basis this increase in accuracy also results in large increases in computation time per opertion. To get a feeling for the numbers involved let us represent a workspace of one cubic feet with domains 1.0 inches on a side. This requires storing 1728 domains. Cutting the domain size to a more reasonable .01 inch. brings the number up by a million to 1,728,000,000 domains.

One extension of the straightforward array representation is the hierarchical array [Nilsson 1969]. Instead of a single predefined grid, this method allows subdividing any rectangular domain into 4x4 grids recursively. Homogeneous domains are not subdivided. Subdivision is only needed at object boundaries. This representation improves on simple arrays by its ability

to represent the large homogeneous areas efficiently.

Eastman [Eastman] has performed some comparisons between several representations for 2-D space. For the figure he used in his paper a simple array requires 1089 domains. A hierarchic array needs only 411. This is a significant improvement but it is possible to do better. Eastman describes another scheme which he credits to Moran [Moran] which requires only 65 domains. This is for the string (list) representation.

The idea behind the list representation is to allow a single internal domain to represent an arbitrary rectangular external domain. An example in two dimensions can be seen in Fig. A2.1. The vertical sides of each rectangular entry defines a "row" in the representation. We need only store the start and end coordinates of this row in a list. Associated with this row is a list of internal domains specified by storing the vertical coordinates of their top and bottom sides. Fig A2.2 shows how adding a new external domain can fragment the representation of a previously stored domain.

The tabular representation for the addresses allows us to store external domains aligned with the axes at an arbitrary accuracy. Non-aligned domains must still be approximated by a number of aligned external domains. The number of aligned domains necessary to represent a non-aligned domain at a specified accuracy increases rapidly as a function of the accuracy. This is true for any space filling representation. The cost of increased accuracy in the list representation is the extra fragmentation. This phenomena is fairly local to objects and does not require an uniform overhead for all objects as in the simple arrays.

The addressing function is fairly simple. It consists of a search down the address lists for the largest address bigger than the first coordinate (X) of the point. At this address is stored another address list which can be searched for the second coordinate (Y). For a 3-D

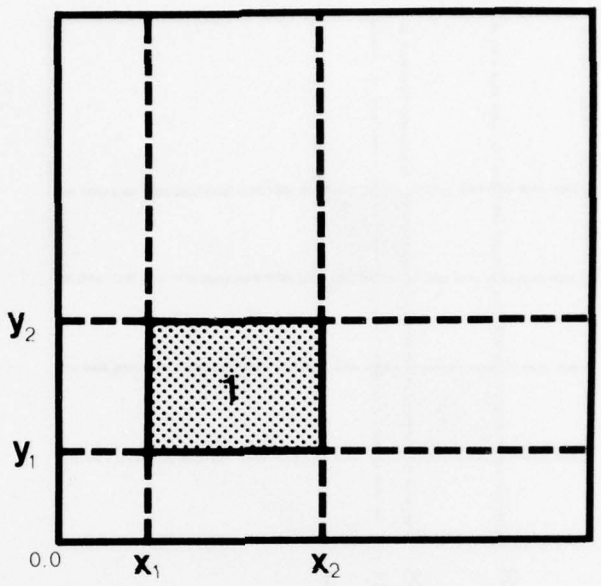


Figure A1.1 - Representation for one object:

```
((0.0  (0.0  0))
 (x1   (0.0  0)  (y1  1)  (y2  0))
 (x2   (0.0  0)))
```

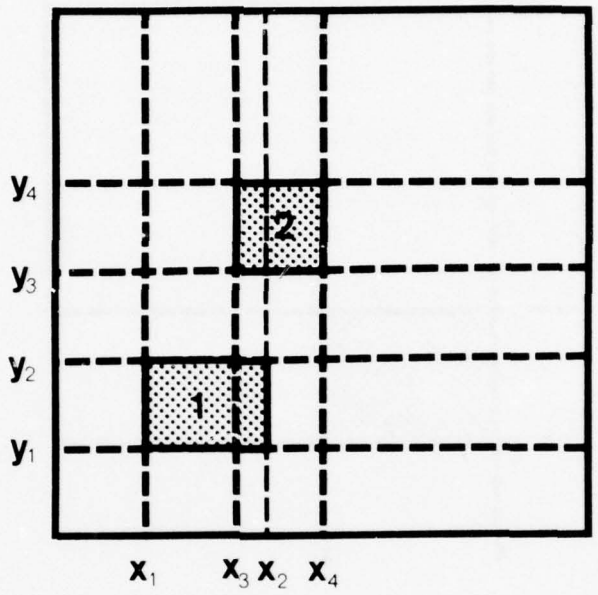


Figure A1.2 - Two objects:

```
((0.0  (0.0  0))
 (x1   (0.0  0)  (y1  1)  (y2  0))
 (x3   (0.0  0)  (y1  1)  (y2  0)  (y3  2)  (y4  0))
 (x2   (0.0  0)  (y3  2)  (y4  0))
 (x4   (0.0  0)  (0.0 0)))
```

representation the process is repeated. At the last coordinate is stored the contents of the domain. A domain pointer in this representation is a list of the three address lists such that the origins of the domain can be read off the first elements of these lists. The contents of the domain is to be found in the last element of the domain pointer.

Inserting a domain into the representation is fairly easy. The following operation is performed for each coordinate. Find the entry in the address list that includes the min value for the coordinate (i.e. the value of the coordinate in the origin of the domain). If the entry in the address list does not equal the coordinate value then the contents of the entry in the address list are copied and the address changed to the coordinate value of the domain. This is the process of splitting a "row". The insertion process is then iterarated with each entry in the address list between the one that includes the origin coordinate and the one including the end coordinate. The splitting is then done for the end coordinate as well.

A2.2. Filling the Space

An important part of a space filling system is the algorithm to generate the domains which are occupied by an object. The basic internal domains are arbitrary rectangular solids not on any predefined grid (variable domains). The goal is to generate as few domains as possible while maintaining small error. The algorithm presented here is meant to suggest the problems involved. It is probably not optimal either in space, time, or simplicity.

The basic idea behind the algorithm is to examine the domains defined by the break points in the lines of the polyhedron as drawn by a 3D vector generator. These domains are tested for inclusion. The vector generator works on a grid whose resolution is determined by the maximum error allowed in the representation. The vector generator generates a list of grid points for each line of the solid. Consecutive points on this list can differ in one, two or three

coordinates. As long as two coordinates are the same the points are considered in line. When two or more differ it is considered a breakpoint. The individual xyz coordinates of these breakpoints determine a grid of variable size domains which are elegible for inclusion in the solid. The next step is to determine which are inside and which outside the solid. This can be done by testing the vertex points of the domains against the planes of the object. The algorithm is roughly as follows:

(1) Generate the breakpoints by "drawing" the lines of the solid.

(2) Generate three list of breakpoints, one for each of x, y and z coordinates of the breakpoints obtained in step 1.

(3) Sort the lists

(4) For each element of the list of x breakpoints do:

(5) For each element of the list of y breakpoints do:

(6) Scan the list of z breakpoints until a point (x y z) is found to be inside the solid. Call this z1.

(7) Scan from z1 until an (x y z) point is found outside the solid. Call this z2.

(8) Store (x, y, z1, z2) and continue.

Each value of x determines a slice through the solid. Each slice is represented by a list of the lists returned by step (8) above. Two such slices for adjacent values of x can then be matched into a "wall" of domains (Fig A2.3).

This method can be improved by using the values of z1 and z2 obtained in one cycle through the z's for a given y as the starting points in the z search for the next y. This focuses the attention of the algorithm at the areas near the plane boundaries. For the first value of y, the program must still examine all the z coordinates. For large number of breakpoints this first scan can be improved by doing a binary search on each of the z's. These two searches can be

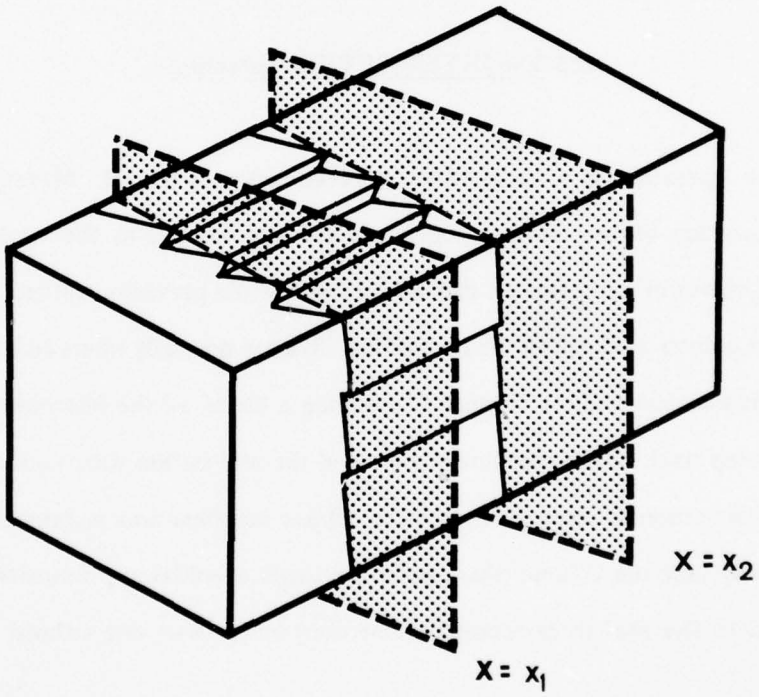


Figure A1.3 - Approximating an object not aligned with the coordinate axis by a series of aligned cuboids.

effectively combined.

For rectangular solids there is an added improvement possible. It is not necessary to generate the breakpoints for the 12 lines of the object. In fact, 6 are sufficient to characterize the volume occupied by the object. These lines can be found by connecting the verteces which have the max and min values for each of the coordinates.

A2.3 The INTERSECTION operation

The intersection operation in a space filling representation is trivial. Merely compute the domain approximation of the object to be intersected and store it in the model. Instead of modifying the contents of each internal domain we check if the previous contents were non-zero; if so the domain pointer is added to the intersection. We are normally interested only in the xyz bounds of the intersection volume. Instead of building a list of all the internal domains in the intersection we keep track of the coordinate bounds of the intersection with each object number. The resulting intersection is returned as a list of object numbers and rectangular intersection volumes. If at any time the volume taken up by the single cuboidal approximation is too large when compared to the real intersection volume then more than one cuboid is used as an approximation.

A2.4. The FINDSPACE problem

The space filling approach does not immediately provide a solution to the FINDSPACE problem. Insofar as it simplifies computing volume intersections it simplifies the verification phase of the FINDSPACE operation. The problem of proposing locations remains. For sparse models the proposer can attempt to find empty internal domains large enough to contain the object we are trying to place. In more crowded models the presence of objects segments most of

the space. This means that empty areas will not generally corresponds to single internal domains. "Region growing" operations can then be used to aggregate empty volume. The problem then is to insure that only convex "regions" can grow.

These problems prompted a different approach to the solution, described in section III.3.2. This method aproximates all the objects, in the area being investigated, by cuboids and then finds all cuboidal areas big enough to fit the target object. This method still requires performing an intersection to determine all the objects in the area of interest.

A major restriction of the method presented in Chapter 3 is that it does not consider rotations of the target object when looking for a place to fit it in a crowded environment. This restriction could be alleviated by using the same type of computation used for positioning the wrist in PGSET[1] (c.f. IV.4.2). We could locate the places where a cube whose dimension is equal to the smallest dimension of the target object would fit. Then the PGSET[1] computation would determine what the set of legal orientations of the target object at that location would be.