LAMA: A LANGUAGE FOR AUTOMATIC MECHANICAL ASSEMBLY

Tomas Loiano-Perer and Patrick H. Winston

Artificial Intelligence Laboratory Massachusetts Institute of Technology Cambridge, Massachusetts 02139

ABSTRACT

This paper describes a mechanical assembly system, LAMA, that transforms high level assembly descriptions into programs for a computer controlled manipulator.

0. INTRODUCTION

Our long-range goal is to create machines that do work of the sort that today requires manipulative, visual, or problem-solving skill not usually associated with assembly automation. We believe that such machines are needed for several reasons:

- To get people out of jobs that \$re dangerous or dehumanizing.
- To get jobs done that people cannot or will not do.
- To reverse the continuing trend toward excessive standardization.
- To compete.

Considerable progress has been made. In many laboratories, robots with sophisticated manipulators assemble devices ranging from bearings to pumps, often working with tolerances requiring force feedback to get things mated.

The programs driving such impressive demonstrations have been straightforward, but tedious to write. The reason is that high level languages are only now under development for automatic assembly. Assembly programming is still in the assembly-language era, so to speak. The programmer is forced to think in terms of instructions like these:

Move in the direction 6 at speed s until you reach position p. Exert a force f by moving in the direction d.

It would be better, and certainly cheaper, if the programmer could think instead in terms of suggestions like these for assembling part of an engine:

Insert the piston pin partway into the piston.

Place the rod's pin end on the piston pin inside the piston.

Push the pin through the rod and the piston.

That is, we would like to tell our robots what to do in terms that •re comfortable to us, leaving them to fill in the details. Rough English description, or something close to English but less ambiguous, is what we have in mind. A mechanical assembly language that works with-such descriptions is on the critical path toward economically viable robot systems.

Our short-range goal, then, is to do something about creating such a language. Indeed, this paper is about LAMA, an acronym for Language for Automatic Mechanical Assembly. It consists of a sample of the work described more fully in a new thesis by the first author [Lozano-Perez] on topics suggested by the second. The translation between what the human foreman-programmer describes and what the manipulator controller wants, in our design, is to take place in two steps:

- First, the human's Assembly Description, with many missing details, is converted into an Assembly Plan, which has the gaps filled in, but remains very high level. A very verbose and careful human's assembly deception would differ only ·lightly or not at all from the assembly plan.
- Second, the Assembly Plan is converted into a Manipulator Program, with things worked out to the level of coordinates, trajectories, forces, and torques. This conversion requires, in part, a pick and place phase and a feedback strategy phase.

The pick and place phase converts the assembly plan into a program that assumes ideal position information and positioning accuracy. It must specify the manipulator motions that achieve the desired relationships between the parts. Both of these assumptions are untenable. The role of feedback planning is to expand skeleton programs embodying feedback strategies to carry out the assembly operations taking into account the imperfection of the data and the positioning errors in the manipulator. The resulting program is the desired manipulator program.

The pick-and-place and feedback-strategy phases have been examined in detail and prototype programs have been implemented that do these jobs. In this paper, we will focus on the phase dealing with feedback strategy.

We begin, however, with a survey of some of the decisions that must be made when specifying an automatic mechanical assembly in general, and we also give a more complete overview of the design of an overall system. We will see that three basic themes underlie the design:

High level assembly operations can be represented by program plans (called skeletons) which can be expanded as required by using facts about the specific task at hand.

- The desired effect of the basic manipulator motions can be described in terms of a few geometrical and spatial relations. We believe that the assembly problem can be seen as the problem of achieving a certain set of geometrical relations between objects while avoiding unwanted collisions.
- Choices for location and motion parameters should be made by identifying all the constraints on the solution, finding a range of values in which the solution may lie and picking an element from that range.

1. THE NATURE OF THE PROBLEM

This section tries to highlight the difficulties of programming assembly operations by examining a particular assembly task in some detail.

Fig. 1 shows the piston subassembly from a model aircraft engine. We will use it to emphasize the number and complexity of the decisions to be made in planning an assembly. The assembly has been carried out using the Silver roboi manipulator 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 ere divided in the following manner: (I) em xy table, (2) a wrist which can displace and rotate along the z axis and (3) a vise which rotates about the x axis (see Fig. 2). The manipulator is equipped with . 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 Silver Robot System to perform precise assembly tasks whose critical clearances are below its positional accuracy [Inoue]

The hard part of this engine subassembly 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 the pin is inserted partway into the piston. Then the piston-rod's pin-end can be placed onto the pin inside the piston. And finally, the pin pushed through the rod and the piston-hole.

This description of the piston assembly 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. Note 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.

The assembly can be carried out by first placing the pin in the vise such that its main axis is horizontal (Fig. 3a). Then the piston can be placed on the pin so that the cavity is facing upward. The rod can then be placed on the portion of pin that is projecting into the piston's cavity (Fig. 3b). The manipulator can then grasp the piston, remove the piston from the vise, and after closing the vise, push the part of the pin protruding from the piston against the vise (Figs. 3c&d). We call this the assembly plan.

For each operation in the assembly plan, the *manipulator* program must specify the position and orientation of the fingers such that they can securely grasp some part of the object to be moved. The program must also specify a trajectory for the manipulator that is not likely to damage the manipulator or disturb any of the other parts.

The assembly description and the assembly plan mention only the spatial relationships that must hold between the parts in the goal state. These relationships involve much greater positional accuracy than is directly achievable by the manipulator to be used. These relationships can be achieved by employing force feedback techniques such as those described in [Inoue].

Inoue has developed a program which, driven by force feedback, does peg-in-hole insertion. Unfortunately, his program cannot be used directly in the task at hand. Inoue's program assumes that the motions of the hand correspond directly to motions of the hole. The assembly plan described above specifies that the piston-rod be held 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 to account for this.

Like most programs, Inoue's has parameters. The length of certain motions and the magnitude of some forces are not specified. These parameters must be specified 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. This can give rise to severe errors.

These considerations point to one, very important, conclusion: Inoue's program is not so much a general utility program as a specification of an assembly strategy to be adapted to *many* different geometric environments. This is true of most manipulator programs because of their dependence on assumptions about the geometry of the task.

Let us summarize the problems to be faced in specifying an assembly operation.

- Describe the objects to be assembled.
- 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.
- Determine the grasping position and orientation for the objects involved in each operation.
- Determine a collision free path between the origin and destination of all motions.
- Tailor the assembly strategies to fit the particular geometric

- environment. This involves providing parameters to the strategies.
- Examine the strategies for likely errors and determine the actions to be taken in case of failure.

2. THE NATURE OF THE APPROACH

The current implementation of LAMA has focused on a subset of the problems outlined in the previous section. The most important are grasping, tailoring assembly strategies and error prediction. This section briefly outlines our approach to these problems.

Computing a grasp point on an object from a description of the object and its surrounding environment is a very difficult computational problem. The basic mechanism needed to solve the problem is the ability to detect that attempting to grasp some part of the object will cause a collision. This involves computing the geometric intersection of the solids making up the manipulator description with the object to be grasped and with any nearby object. A non-null intersection indicates a collision. A simple approach to solving this problem would be to guess a grasp point and then test whether it is safe. Typically there is an infinite number of possibilities and the challenge is to choose a grasp point with some confidence that it is a good choice. Trial and error methods are unsatisfactory in this context because not only do they have very poor worst-case behavior but once a result is found, it is very difficult to evaluate it without knowing the alternatives. Our approach to grasping, briefly described in section 3, relies on describing the range of possible grasp points as a few sets of parameterized grasp positions which can be evaluated and compared as units.

In section 1, we pointed out the dependence of manipulator programs on the particular geometry they were designed for. A useful mechanical assembly system must have some means of representing assembly strategies that does not make too many assumptions about the environment where they are to be used. Our approach to making the strategies as general as possible is to have each of the steps in the strategy be essentially a goal statement. Each operation is decribed in terms of the geometric relations it is meant to achieve in a coordinate system specified by the strategy. The assembly strategies can then be adapted to a particular situation by computing the parameters needed to achieve the goal of the individual steps of the strategy.

Once we have available a description of the goal of each step, then some errors can be predicted by simulating the operations, taking into account some of the uncertainty in the positions of objects. These error predictions can be used to solicit corrective actions from the user.

Section 3 will describe in more detail how the assembly strategies are used.

3 AN OVERVIEW OF THE LAMA SYSTEM

This section describes how we believe the piston assembly will be processed by LAMA when it is completely implemented. Prototype programs currently exist to do parts description, the Pick and Place phase and a simple Feedback Strategy phase. No program exists to do the Assembly Planning.

The parts to be assembled must first be described to the system. The user uses the system interactively to define models of the parts. Complex objects are described as unions of a few kinds of primitive object types. The primitive objects currently available in implemented parts of LAMA are a cuboid and a cylinder (both as solids and holes).

Fig. 4 shows a schematic description of the models for the parts in the piston assembly. Note 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. 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. By generalizing to polyhedra the desired volumes can be approximated 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. A more realistic goal is that of accepting assembly instructions similar to those given to people, as discussed in the introduction.

The assembly description for the piston assembly is shown in Fig. 5. The assembly description s-pecifies only that operations, such as insertion, are to be performed on the parts. The individual operations ara often under-specified. Paramelers •re 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.

The first 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 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 note 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. 5 is shown in Fig. 6.

The current implementation assumes that the Assembly Plan is directly available as an input to the system. We can then focus on the process of transforming an assembly plan into a manipulator program.

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. 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. 7 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 rest of the environment.

Fig. 8 shows the system's representation for the peg-in-hoU insertion strategy. It is very similar to the program presented in [Inoue]. The Feedback Strategy phase simulates this skeleton program, predicts contacts and estimates the direction and magnitude of the forces that will be produced.

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

- 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 geometrical relations it is meant to achieve
- 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 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. A test for this situation can be generated and instructions as to corrective action could be solicited from the user.

4. THE FEEDBACK STRATEGY PHASE

The result of the operation of the Feedback Strategy phase is a manipulator program. This section considers the operation of the Feedback Strategy phase 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. A preliminary implementation currently exists of the program that does the simple code generation shown here.

The assembly plan (Fig. 6) has the following entries for the operation of inserting the piston-rod on the piston-pin:

(GRASP OBJ : [PISTON-ROD]

SUCH-THAT : (FACING* ([ROD-BAR] TOP) UP))

(INSERT 0BJ1 : [PISTON-PIN]

0BJ2 : [PISTON-ROD PIN-END-HOLE])

(UNGRASP OBJ : [PISTON-ROD])

We must first find a grasp point on the piston-rod. There are two possible grasp positions on the rod; 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 and (2) possible collisions. Collisions are predicted by simulating the assembly operations while assuming the hand can be at both of the legal grasp positions. The Feedback Strategy phase can then consider the effects on each of the grasp ranges simultaneously.

The first task in expanding an assembly strategy is to setup the local reference system. The REFERENCE statement in PEG-IN-HOLE 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, the reference's z is aligned to the global r.

The INITIAL statement specifies the constraints on the initial position of the parts. In PEG-IN-HOLE it specifies that the HOLE and the PEG be ALIGNED&CENTERED and IN-FRONT-OF each other. Fig. 9a shows a top view of the interaction volume

of the piston-rod's pin 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.

The first step in the DROP-INTO strategy calls for the object in the hand to be rotated 0.1 radians. The Feedback Strategy phase must establish that this rotation will not have any bad effects. This is done by simulating the motion. In this case, contacts with the pin and/or the piston are possible. These accidental contacts determine that the force parameter be "dtttct"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 0.1 WHILE (RFORCE < "defect-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. 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 pm. 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. 9b shows (in dashed lines) the volume taken up by the piston-rod over the range of positions consistent with this relation.
- (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. 9b shows (in solid lines) the volume of the rod over the range of positions consistent with this relation.

A position consistent with both of these relations is obtained by computing the range of values of the position parameters that satisfy each one and then intersecting the "ranges. Fig. 9b shows a graphical representation of the ranges and their intersection.

A simulation of the motions shows that the rod can come in contact with the pin and with the inside of the piston (Fig. 9c). 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 landing step is simulated. 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 V WHILE (XFORCE < "detect-contact"))
(COND ((CONTACT? X 0.0 V)) (T (ERROR)))
This completes the DROP-INTO operation. The complete program can be seen in Fig. 10.

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. 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.

5. RELATION TO OTHER WORK

The problem of construction planning has been very important 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 have explored general issues of planning and debugging in the context of assembly problems. [Sussman] treated assemblies of blocks exclusively while [Sacerdoti] 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 block structures could ignore most of the problems of spatial interactions. BUILD also ignored the manipulation aspect of the construction. Sacerdoti's use of NOAH in SRI's Consultant project avoids alt these problems because it assumes a human as the manipulator. The work on LAMA has focused on the problems introduced by more realistic objects and the errors of a real manipulator.

The approach taken in LAMA to assembly strategies was influenced principally by the work of Sussman [Sussman] and Goldstein [Goldstein] in debugging programs. They both stressed the usefulness of having a statement of purpose for each operation in a program. The elegant method of [Ambler & Popplestone] for computing the position and orientation of objects given relations such as AGAINST and FITS-IN provided some of the key ideas on how to describe the goals of strategy steps.

There are, at least, five other projects that have direct relevance to the task of building an automatic mechanical system. These projects are being conducted by the IBM Thomas J. Watson Research Center, the Stanford Al Laboratory, the Stanford Research Institute, the University of Edinburgh, and the C. S. Draper Laboratory.

The IBM system design, AUTOPASS [Lieberman & Wesley], is closest to LAMA. 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 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 Edinburgh group [Ambler et. al.] has focused on the problem of assembling an object whose parts must be visually

located and pulled out of a heap. An early speculative paper from their group [Popplestone] anticipated many of the ideas and approaches adopted in this research, even to the choice of a model aircraft engine as the example.

The Draper Lab [Nevins et. al.] 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.

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.

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.

[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.
[Goldstein]

LP. Goldstein, *Understanding Simple Picture Programs*, MIT Artificial Intelligence Laboratory Technical Report 294, September 1974.

[Inoue]

K Inoue, Force Feedback in Precise Assembly Tasks, MIT Artificial Intelligence Laboratory Memo 308, August 1974.

[Lieberman & Wesley]

L 1. Lieberman and M. A. Wesley, AVTOPASS, A Very High Level Programming Language for Mechanical Assembler System, IBM Research Report RC-5599, August 1975.

[Lozano-Perez]

T. Lozano-Perez, *The Design of a Mechanical Assembly System,* MIT Artificial Intelligence Laboratory Technical Report 397, December 1976.

[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.

[Popplestone]

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

[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]

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

[Sussman]

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

[Taylor]

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

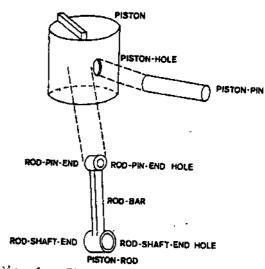


Fig. 1 - The piston sub-assembly

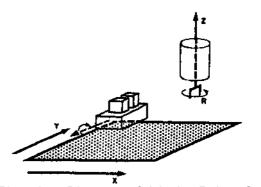


Fig. 2 - Diagram of Little Robot System, see (Silver).

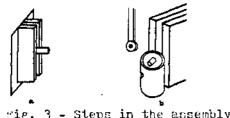
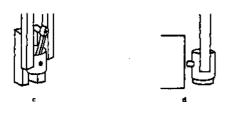
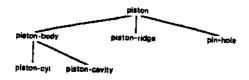


Fig. 3 - Steps in the assembly of the piston.





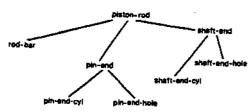


Fig. 4 - Parts descriptions for piston sub-assembly,

(INSERT OBJI: [PISTON-PIN] OBJ2: [PISTON PIN-HOLE]

SUCH-THAT: (PARTLY (FITS-IN OBJI OBJ2)))

(INSERT OBJI: [PISTON-PIN]

OBJ2: [ROD SHALL-END-HOLE])

(PUSH-INTO OBJI: [PISTON-PIN]

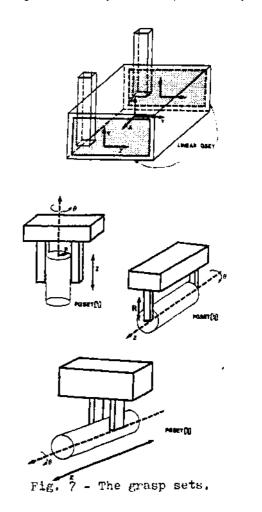
OBJ2: (AND [PISTON PIN-HOLE] [PISTON-ROD SMALL-END]))

Figure 5. - Initial Assembly Description for the piston assembly.

(GRASP OBJ: [PISTON-PIN]) (PLACE-IN-VISE OBJ: [PISTON-PIN] SUCH-THAT: (PARALLEL [PISTON-PIN] [TABLE])) (UNGRASP OBJ: [PISTON-PIN]) (GRASP OBJ: [PISTON] SUCH-THAT: (FACING+ ([PISTON] TOP) DOWN)) (INSERT OBJI: [PISTON-PIN] OBJ2: [PISTON PIN-HOLE] SUCH-THAT: (PARTLY (FITS-IN OBJI OBJ2) 0.25)) (UNGRASP OBJ: [PISTON]) (GRASP OBJ: [PISTON-ROD] SUCH-THAT: (FACING+ ([ROD-BAR] TOP) UP)) (INSERT OBJI: [PISTON-PIN] OBJ2: [PISTON-ROD SHALL-END-HOLE]) (UNGRASP OBJ: [PISTON-ROD]) (GRASP OBJ: [PISTON]) (REMOVE-FROH-VISE OBJ: [PISTON]) (PUSH-INTO OBJ: [PISTON-PIN] SUCH-THAT: (AND (FITS-IN [PISTON-PIN] [PISTON PIN-HOLE]) (FITS-IN [PISTON-PIN] [PISTON-ROD SMALL-END])))

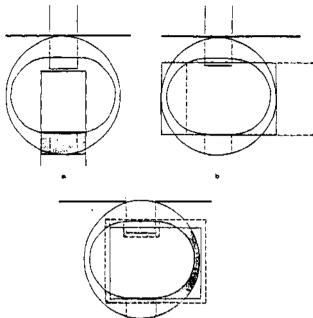
(UNGRASP OBJ: [PISTON])

Figure 6. - Assembly Plan for the piston assembly



STRATEGY	<u> </u>
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))) (HATE: (HATE 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 0.1) SUCH-THAT (ALMOST (ALIGNED- PEG HOLE) 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)	ig. 9 pi
SUCH-THAT (AND (ABOVE (PEG CENTER) (HOLE CENTER)) (CONTACT PEG (HOLE SIDE)))) (SAVE1 : (SETQ ZI ZPOS)) (EDGE- : (CHANGE Z) SUCH-THAT (AND (BELOW (PEG CENTER) (HOLE CENTER))	
(CONTACT PEG (HOLE SIDE)))) (SAVEZ : (SETQ Z2 ZPOS))	(STRATEGY DROP-IN (ROTATE
(CENTER: (HOVE Z) SUCH-THAT (BETWEEN (PEG CENTER) ZI Z2)) (CONTACT: (CHANGE Y) SUCH-THAT (CONTACT PEG (HOLE SIDE)))	SUCH-T (SHIFT SUCH-T
(HATE : (CHANGE R WITH (AND (ZFORCE = 0.) (YFORCE = "NAINTAIN-CONTACT"))	(LANDIN SUCH-T
SUCH-THAT (ALIGNED- PEG HOLE)))	(DEFUN DROP-IN
(STRATEGY	(CHANGE (cord (
PUSH-INTO (PEG HOLE) (PUSH : (CHANGE X WITH (AND (YFORCE = 0.)	(CHANGE
(ZFORCE = 0.))) SUCH-THAT (FITS-IN PEG HOLE)))	(cord ((changi (cord (

Figure 8. - Peg-in-hole strategy. The representation of Inoue's peg-in-hole insertion strategy in LAMA.



ig. 9 - The position ranges for the piston-rod insertion.

```
ROP-INTO (PEG HOLE)
ROTATE : (CHANGE R BY 0.1)
SUCH-THAT (ALMOST (ALIGNED- PEG HOLE) 0.1))
SHIFT : (CHANGE Y)
SUCH-THAT (LEFT-OF (PEG CENTER) (HOLE CENTER)))
LANDING : (CHANGE X)
SUCH-THAT (CONTACT (PEG FRONT) (HOLE FRONT))))
ROP-INTO (PEG HOLE)
(CHANGE R BY 0.1 while (rforce ( "detect-contect"))
cond ((contact?) (error?)))
CHANGE Y by "y" while (uforce < "detect-contact"))
cond ((contact?) (error?)))
(CHANGE X by "x" while (xforce ( "detect-contact"))
cond ((contact?)) (t (error?))))
```

Figure 10. - DROP-INTO strategy and its expansion into LAMA. The text in italics indicates the parts generated by the Feedback Strategy phase