A Machine Learning Based Logic Branching Algorithm for Automated Assembly

by

Erik Garth Vaaler

S.B. University of California at Berkeley
(1976)

S.M. Massachusetts Institute of Technology
(1984)

Submitted to the
Department of Mechanical Engineering
in Partial Fulfillment of the Requirements for the Degree of

Doctor Of Science

at the
Massachusetts Institute Of Technology
January 1, 1991

Signature of Author________________________________

Erik Garth Vaaler
Department of Mechanical Engineering
January 1, 1991

Certified by [illegible]

Professor Warren P. Seering
Thesis Supervisor

Accepted by________________________________

Ain A. Sonin
Chairman, Departmental Graduate Committee

A Machine Learning Based Logic Branching Algorithm for Automated Assembly

by

Erik Garth Vaaler

Submitted to the Department of Mechanical Engineering on January 1 1991, in partial fulfillment of the requirements for the degree of Doctor of Science in Mechanical Engineering.

Abstract

A primary source of difficulty in automated assembly is the uncertainty in the relative position of the parts being assembled. This thesis focuses on a machine learning approach embedded in a logic branching structure to accomodate this uncertainty in peg and hole assemblies. Force sensor information, responses to recent moves, and results from previous assemblies are used as sources of information for the learning algorithm. Machine learning is used to to generate the branching decisions (production rules). Several heuristic assembly algorithms are developed and tested both in a computer simulation and on a real system.

Thesis Committee:	Dr. J. Kenneth Salisbury
	Prof. Warren P. Seering, Chairman
	Prof. Thomas B. Sheridan

Acknowledgments

Many people contributed to the work described in this thesis. In particular: my thesis advisor, Warren Seering; members of my thesis committee, Tom Sheridan and Ken Salisbury; and the past and present members of my research group: Al Ward, Rob Podoloff, Mike Benjamin, Mike Caine, Andy Christian, Steve Eppinger, Steve Gordon, Peter Meckl, Ken Pasch, Neil Singer, Karl Ulrich, Kamala Sundaram, Jim Hyde, Brian Avery, Lukas Rueker, Bruce Thompson, and Bill Singhose (yes, I was here a long time). Special thanks to The Captain.

This research was performed at the MIT Artificial Intelligence Laboratory and funded by a Weyerhauser fellowship, URI grant # N00014-86-K-0685, and SDF grant # G246. I would also like to acknowledge that thanks to the faculty, staff, and students, the MIT Artificial Intelligence Laboratory is a very supportive research environment.

Contents

List of Figures

Chapter 1

Introduction

1.1 Thesis Overview

1.1.1 General Problem Statement

For most applications, robotic assembly cells with the accuracy necessary to successfully assemble parts using open loop path follcwing are far too expensive to compete with human assembly workers. Possible solutions to this problem include: improving the cost to performance ratio of the robot, redesigning the parts so that they can be assembled more easily, and installing sensors on the robot and fixtures to gain access to information about the assembly that can be used to correct for the non-ideal behavior of the robot. The latter approach is the one taken in this thesis.

1.1.2 Goals

The principal goal of this work was to explore the feasibility of peg and hole assembly using machine learning to learn the correct responses to contact forces encountered during assembly. A secondary goal was to determine the practical limits of the accuracy of the mapping from contact forces to relative part positions.

1.1.3 Motivation

For most companies, assembly accounts for more than 50% of the cost of manufacturing a product. For complex machines manufactured in small quantities, the percentage can be much higher. Reducing the cost of assembly, particularly for small production runs, is a primary motivation for research in automated assembly. Another motivation for work in this area is the increasing need for work in extremely hazardous environments, in particular: nuclear reactors, offshore drilling platforms, undersea pipelines, and space station and satellite construction and repair.

1.1.4 Scope

The work in this thesis is limited to the development of algorithms for the assembly of two rigid, smooth parts (modeled as a planar peg and hole), one with known orientation and the other with an orientation known to within ± 5 degrees. The work assumes that the peg has been located over the hole by some other method, e.g. open loop position, vision, or tilt-and-drag.

1.1.5 Description of the Approach

The underlying structure to the approach presented in this thesis called logic branching [Whitney 85]. Logic branching is a discrete approach to controlling systems. Logic branching routines generally have the same structure as the IF-THEN-ELSE, DO-WHILE, or DO-UNTIL statements have in computer programming. For assembly applications, the arguments to the conditional part of the statement (the branch points) are usually force conditions, e.g. DO (move in X) UNTIL (Fx $>$ 5). Force sensors, responses to recent moves, results from previous assemblies, off-line calculations, and simulation results are some of the sources of information that can be used to generate the branching decisions. The decisions typically result in a move or series of moves that attempts to orient the parts so that a particular set of contact forces is present. Often this is simply an attempt to reduce contact forces under a certain threshold that will allow the system to successfully continue the assembly.

Most software approaches to automated assembly produce algorithms that are static and deterministic. The response to sensor information, if any sensors are used, is explicitly set by the programmer. My work differs from this approach in a fundamental way. It is a very simple form of machine learning. Here, a program is written that "learns" the desirable responses to sensor inputs. An advantage learning algorithms can have over analytical solutions to a problem is the ability to correctly handle unexpected data. For example, an analysis of the peg and hole geometry will show that there are combinations of forces and torques that can not occur. All of the responses to these force/torque combinations could be thrown out, and often are, since it is difficult to generate anything beyond an error message in response to a combination of forces that a model says is impossible. Modeling errors generally result in the real system encountering some of the force combinations that the model has deemed to be impossible. On the other hand, a learning algorithm can be set up so that no *a priori* decisions need to be made about the possibility of certain force combinations. The system simply does not ever encounter the impossible combinations. Unsupervised learning was used because I felt that it produces a more robust system and a clearer picture of the effect of the underlying structure of the learning algorithm than either supervised learning or learning from examples. However, a real world application would certainly benefit from these other forms of learning.

Sufficient information is available from the force sensor, responses to recent moves, and results from previous assemblies so that the approach does not require a 3 dimensional model of the assembly process, i.e. the part and robot geometries, kinematics, stiffnesses, etc.. The only information about the assembly that is required is a nominal assembly path (NAP) and a termination condition. The NAP is the path that would be followed if the assembly was attempted open loop. For the common Z-stack assembly the NAP is simply a straight line parallel to the Z axis and coincident with axis of the hole. I used a standard termination condition: a force threshold along the NAP combined with a sensed position indicating that the parts are in the neighborhood of being assembled. Position errors in the system that make open loop assembly impossible are corrected by the assembly algorithm. Force and incremental move data is collected during each assembly and is used by the assembly algorithm during future assembly attempts to improve the behavior of the system by encouraging the peg to move toward the NAP.

1.2 Clarification of Terminology

This thesis contains some terminology that should be clearly defined at the outset, either because it's use in technical papers has not been completely consistent or because I coined it for use in my work.

Although the assembly algorithms do not use cartesian coordinates, I often use them to describe the system errors and motions. I generally break them down into translational (X,Y,Z) and rotational (Θ,Φ,Ψ) terms. Torques on the rotational degrees of freedom are called forces, e.g. F_θ, to avoid confusion with the Tilt Axis on the robot. The Tilt Axis, usually abbreviated to 'T axis', is used for motion in the Θ direction. I have borrowed the concept of *state* from control theory to describe the relationship between the parts being assembled. A complete set of states is any set of system parameters that unambiguously defines some aspect of the system behavior. I use *to exist in a state* and *to visit a state* interchangeably when describing the robot/part system. The system behavior I want to control is the relative position of the parts being assembled. To do this, I use a set of force measurements and calculations using recent incremental moves made by the robot. I have called some of these states *force derivatives*. The force derivative states are the ratios of the change in measured force resulting from an incremental move by the robot to that incremental move, e.g. $\Delta Fx/\Delta Z$. I call the acquisition of state information a *state measurement*. This may be simply a measurement of force or Z position or it may be the calculation of a force derivative. The state information is stored in an array for the simulation (for ease of inspection) and in a list for the real hardware (for compact size). Each entry in the array/list has a set of indices associated with it. These indices, called *state values*, are discretized, normalized state measurements. Figure 1.1 shows an example of this transformation.

I created the acronym NAP (Nominal Assembly Path) that has the following definition: the NAP is the assembly path that the robot would follow (successfully) if the assembly cell and the parts were perfect, i.e. zero position errors in the robot and fixtures and parts made to zero tolerance.

The work presented in this thesis was implemented on the MIT Precision Assembly Robot (MITPAR) which is shown in Figure 1.2. Design and construction details for this machine are presented in Appendix A.

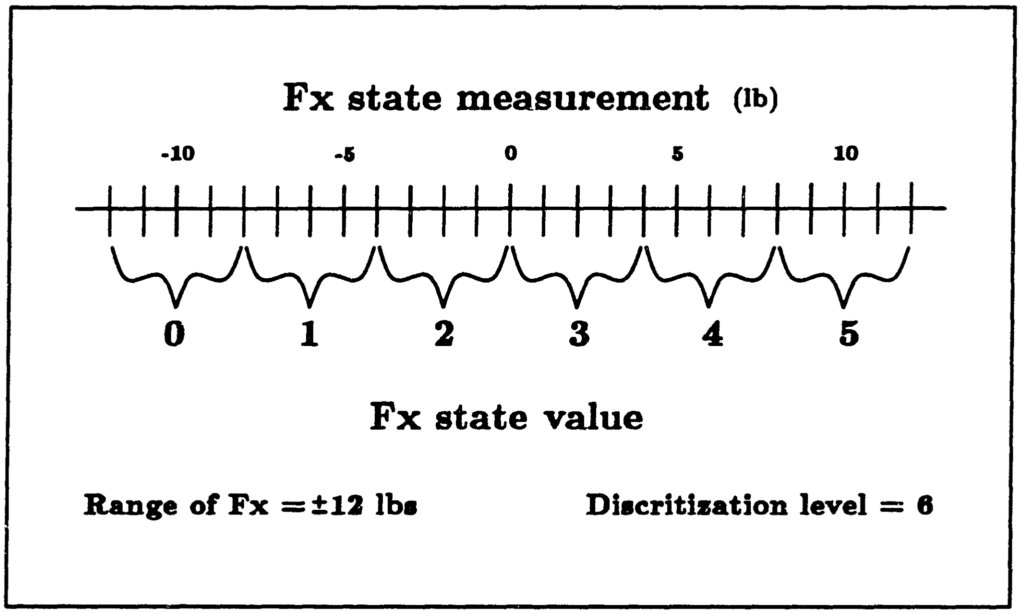

Figure 1.1: The Transformation of State Measurements to State Values - The range of the state measurements (±12 lbs) is normalized by mapping onto the range of the state values (0 through 5). An analog force measurement is mapped into one of the 6 discrete regions.

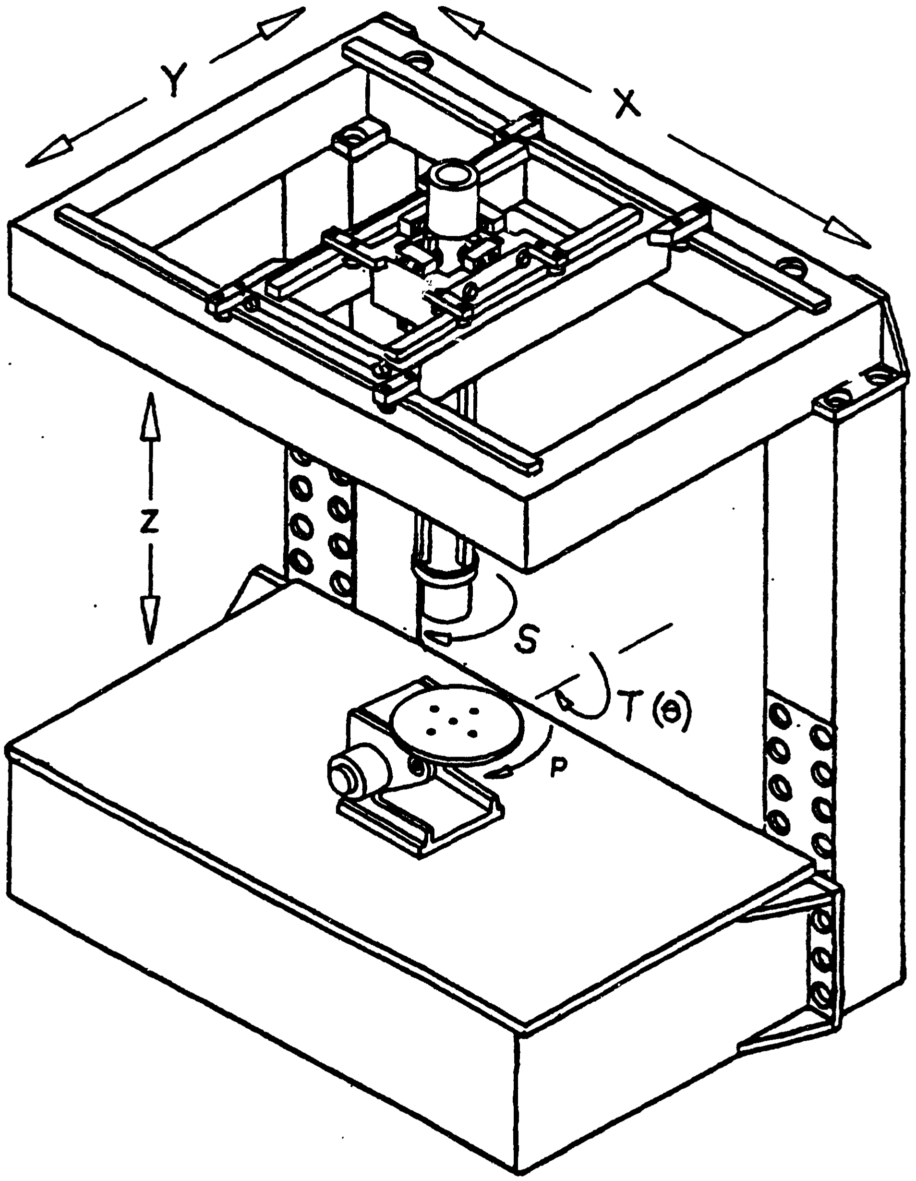

Figure 1.2: The MIT Precision Assembly Robot (MITPAR) - The overall size of the machine is 48 inches by 32 inches by 72 inches (X, Y, Z). The workspace is approximately 24 inches by 12 inches by 18 inches (X, Y, Z).

1.3 Review of Automated Assembly Techniques

1.3.1 DFA - Design for Assembly

DFA is an attempt to design parts so that they can be assembled in the presence of system position errors as well as errors in the parts themselves [Boothroyd 80]. This area of research is mentioned here because the design of a part can have a strong impact on the success of a particular assembly strategy. This coupling has become more of an issue with the increase in interest in DFM (Design For Manufacturing) because the application of DFM and DFA to a part often result in conflicting design specifications.

A very successful DFA technique has been to chamfer the leading edge of mating parts. Although simply cutting or molding a 45 degree chamfer into parts is often done and generally results in increased assembly success rates, the optimal size and shape of a chamfer is a function of many part and robot system parameters. Chamfers and other DFA techniques are the subject of considerable research [Whitney, Gustavson, Hennessey 83, Miller 88, Caine 90].

1.3.2 Part Locating Systems

Part locating systems typically use some type of sensor (often a camera) to determine the location of a part relative to the gripper or the other part [Grimson and Lozano-Perez 83, Grimson 85]. [Gordon 86] contains a substantial literature review of the topic. A common application is the assembly of printed circuit boards. The robot picks up an electronic component, briefly holds it in front of a camera or cameras, and the difference between the actual and desired part position is calculated and corrected. Manufacturing errors and damage can also be checked for at this time. The relative position of the parts is then known accurately enough so that the actual assembly can be performed open loop.

Another method for locating parts is to make the parts contact each other in several places. The robot position during each of these contacts is then run through an algorithm that produces the desired relative position information [Simunovic 79, Gaston and Lozano-Perez 83, Grimson and Lozano-Perez 83]. [Schneiter 86] developed an algorithm that will generate the points where the parts should be made to contact in order to reduce the relative position uncertainty to a given value using the minimum number of contacts (and time). A more passive approach uses a probabilistic estimator/filter to predict the likeliest position and orientation of the parts based on data from previous assemblies, e.g. an RCC with position sensors [Johnson and Hill 85, Seltzer 82].

1.3.3 Force Control - Path Planning

The goals of the work presented in this thesis should not be confused with the goals of most of the research on path planning algorithms. A very significant simplifying assumption was made in my thesis. That is, I assume that there is free space between any position of the peg and hole and the NAP. The scope of this thesis is much narrower than, for example, the path/motion planning work of [Brooks 82a 82b 83, Lozano-Perez 84, Canny 84, Buckley 87]. Path planning algorithms are often applicable to objects with arbitrarily large initial orientation errors moving (without collision) through a space cluttered with arbitrarily many objects. Many of these approaches are therefore suitable for navigation and obstacle avoidance as well as for assembly. Work on path planning with friction and uncertainty [Donald 87, Lozano-Perez, Mason and Taylor 84, Erdmann 84] is particularly relevant to assembly.

1.3.4 Force Control - Compliant Motion

This approach uses the geometric constraints imposed by the interacting parts to guide the parts together[Inoue 74, Van Brussel and Simons 78, Raibert and Craig 81, Whitney 85, Peshkin 88]. Compliant motion assembly can be performed with hardware or software. [Mason 81] develops models of assembly using position control - the *generalized spring*, and velocity control - *the generalized damper.* For any particular peg and hole geometry there is a combination of center of rotation and translational and rotational stiffness that minimizes the insertion forces. [Salisbury 80] presents an approach to implementing the desired stiffnesses and centers of rotation in software. The RCC (Remote Center of Compliance) is a mechanical linkage that can be built with a wide range of these parameters [Whitney 82, Whitney and Rourke 86]. This work was extended to polygonal (e.g. rectangular), unchamfered pegs by [Caine 85, Strip 87 88] and to dynamic applications by [Asada and Kakumoto 88]. The RCC has several limitations. Any particular set of stiffness and rotation parameters only works well for a limited set of part geometries. The RCC requires relatively accurate knowledge of the rotational orientation of the parts being assembled. A possible solution to these problems is the implementation of an RCC or other compliant behavior in software [Hogan 84, Caine 85]. A combination of low system bandwidth (relative to the RCC) and system nonlinearities such as backlash and friction on existing robots has severely limited the performance of software compliance approaches [Eppinger 88]. The development of micromanipulators and highly backdrivable robots [Whitney and Nevins 78, Asada and Youcef-Toumi 87, Townsend 88] should improve the performance of these algorithms significantly.

1.3.5 Force Control - Logic Branching

Logic branching routines generally have the same structure as the IF-THEN-ELSE, DO-WHILE, or DO-UNTIL statements have in computer programming. The arguments to the conditional part of the statement (the branch points) are usually force conditions, e.g. DO (move in X) UNTIL ($Fx > 5$). Force sensors, responses to recent moves, results from previous assemblies, off-line calculations, and simulation results are some of the sources of information that can be used to generate the branching decisions. The decisions typically result in a move or series of moves that attempts to orient the parts so that a particular set of contact forces is present. Often this is simply an attempt to reduce contact forces under a certain threshold that will allow the system to successfully continue the assembly.

Although compliant motion can be implemented with a logic branching structure, the causality of logic branching is typically the opposite of the causality of compliant motion. Given a certain position, a compliant motion routine typically imposes a set of forces on the parts in the assembly which produces relative motion in the parts in the desired direction. Logic branching routines generally establish contact between the parts being assembled and then, based on the measured forces and other information, make a move in the desired direction. Logic branching based assembly can also be viewed as a quasi-static system equivalent of digital feedback control of dynamic systems. The input to either system is a set of sensor outputs sampled at discrete intervals that are monitoring the system behavior. The output is an action that influences the system.

1.3.6 Relevant Aspects of Machine Learning

Although the application of machine learning to mechanical systems is rare [Michie and Chambers 68, Dufay and Latombe 84, Simons, et. al. 85, Connell and Utgoff 87], research on machine learning is a significant part of the overall research effort in Artificial Intelligence [Michalsky, et.al. 83 86 90]. Using the terminology of [Michalsky, et.al. 83], the learning algorithm presented in this thesis is a *production system* that generates rules using *unsupervised learning.* The program contains a function that drives *active experimentation,* i. e. interaction with the environment instigated by the program. The acquired "knowledge" is represented in the form of *production rules.* Production rules are *condition-action pairs.* In this case a corrective move made in response to a particular set of forces, torques, etc.. A machine learning algorithm can generally be divided into 4 components [Smith, et. al. 77] : a Problem Generator, a Performance Element, a Critic, and a Learning Element. The Problem Generator initializes and starts the system. This corresponds directly with INITIALIZE-SYSTEM (block 1, Figure 2.3). The Performance Element is the output of the algorithm and is responsible for generating a control action, in this case a move in X or Θ(block 3, Figure 2.3). The Critic and the Learning Element are embedded in the assembly algorithm (block 6, Figure 2.3). The Critic evaluates the quality of the data and/or selects a subset of "good" data from this set. This evaluation is used by the Learning Element to transform the learned information into a form usable by the Performance Element.

Chapter 2

Formulation of the Assembly Task

2.1 Geometry

Two nearly universal assumptions are made when using peg and hole models: 1) the hole is fixed; 2) one of the parts, usually the hole, is in a known orientation (typically vertical). The first assumption, if incorrect, has no impact on most assembly algorithms because only the *relative* position of the parts is the parameter of concern. The second assumption may severely and unrealistically restrict the set of possible part contact geometries. This can result in overly optimistic estimates of the performance of a particular assembly algorithm. Significant errors in the orientation and the position of both the peg and the hole very rarely occur in assembly cells, but as more robotic assembly and repair work is done in loosely structured environments (undersea, space) the need for assembly algorithms to accommodate position errors in *both* parts will increase. The location of the force sensor, generally assumed to be monitoring forces on the peg, has a significant impact on what force information that can be generated in the general assembly case. For example, when using the standard robot, force sensor, peg, and hole configuration, if the hole has a larger orientation error than the peg relative to the NAP (Figure 2.1), variations in the contact point between the peg and hole (from which the relative Z position of the parts can be determined) can be measured in some cases (a and b) but not in others (c and d). For small errors or large clearance ratios, this condition is not necessarily a problem. For errors so large that the peg can not enter the hole, enough of the error must be in the peg so that 2 point contact inside the hole can be established after the error in the peg is corrected for. I have limited my work in this thesis to assemblies where the orientation of one of the parts is known.

2.2 Mapping Contact Forces into Relative Positions

There is a minimum set of measured system variables that are necessary to determine the state of a system.To determine the relative position of a pair of objects (the state of the system), the 6 cartesian degrees of freedom (X, Y, Z, Θ, Φ, Ψ) are one possible set of states. A possible set of states for the 2-D system used in this thesis is: X, Z, and Θ (shown in Figure 2.1). In an error free system, the cartesian coordinates associated with each part can be used to calculate the relative positions of the parts. Part location errors and robot position errors in real systems limit the accuracy of this information. This is, of course, the source of much of the difficulty of automated assembly. Since the standard cartesian coordinates are not accessible with the necessary accuracy, I use a different set of coordinates.

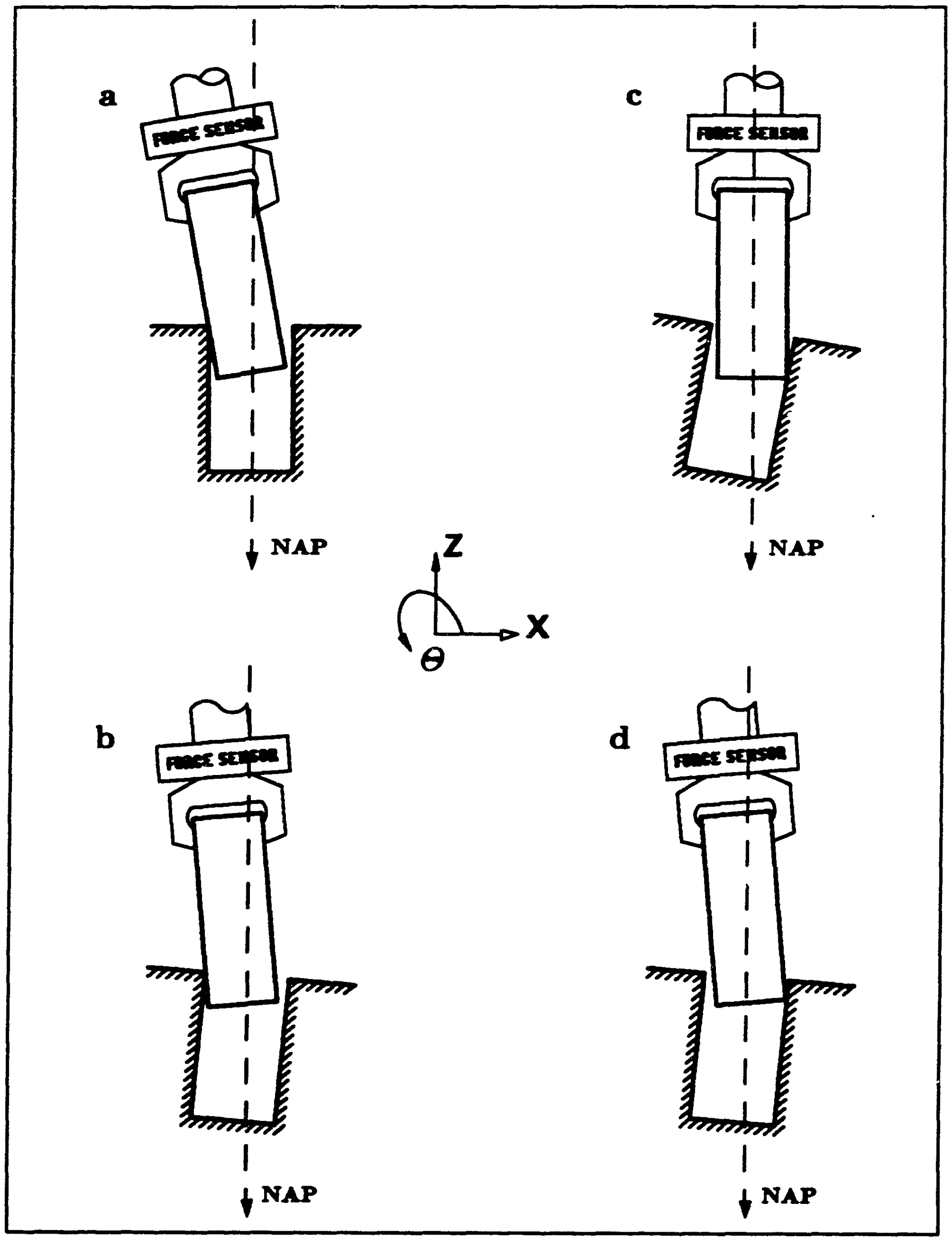

Figure 2.1: Possible Peg and Hole Errors and Contact Configurations - Although there can be a significant angular position error in the peg (a), in the the hole (c), or in both (b and d), only the most common case (a) is addressed in this thesis.

During an assembly, the following information is collected and/or calculated:

1. F_x: the sum of measured forces in the X direction
2. F_z: the sum of measured forces in the Z direction
3. F_θ: the sum of measured forces in the Θ direction
4. $\Delta F_x / \Delta Z$: incremental change in F_x for an incremental change in Z
5. $\Delta F_\theta / \Delta Z$: incremental change in F_θ for an incremental change in Z
6. Z: the measured position of the Z axis

Given the idealized, 2-D assembly case where there are orientation errors in both the peg and hole, perfect part and robot models, and deflections due to contact forces that are small compared to the size of the parts and to the distance from the point of contact to the center of compliance of the robot, there is a transformation from these states to X, Z, and Θ during two point contact. If the orientation of the axis of one of the parts is known, the transformation also exists for single point contact. The latter is typically the case, but even when neither part orientation is accurately known, during one point contact these states contain information that can be used to determine the error, relative to the NAP, of one of the parts (which one depends on the type of contact). Correcting for errors in one of the parts will often be sufficient to assemble the parts beyond the point where wedging or jamming can occur. Two point contact is then "safe", and force information can be collected to correct the remaining misalignment.

There are cross coupling terms due to deflections in the robot and the parts, but to a first order, *during one point contact:*

1. $X_{rel} \propto F_x$
2. $Z_{rel} \propto F_x/T_\theta$
3. $\Theta \propto arctan(\Delta F_x/\Delta Z)$

During one point contact:

- **For small angles of contact (less than 5 degrees or so), or for larger angles of contact where the uncertainty is small (e.g. during a tilt and drag operation where the known angle is 30 degrees with an uncertainty of 5 degrees), the ratio of F_x/F_θ is an accurate measure of the distance from the point of contact to the force/torque sensor.**
- **The angle between the peg and the hole can be determined from the value of $\Delta F_x/\Delta Z$ and the values of K_x and K_θ, the system stiffness in the X and Θ directions. As long as the spring constants are repeatable, the actual values need not be known, because there will then be a consistent relationship between the value of $\Delta F_x/\Delta Z$ and the angle of contact (Θ).**
- **If the relative position of the parts is known in Z and Θ, contact between the parts (with $F_x = 0$) locates the parts in X also. If $F_x > 0$, the relative X position of the parts is related to the force in X and the stiffness of the robot system at the point of contact.**

During two point contact:

- **The relative position of the parts in Z can no longer be determined by F_x/F_θ because the system is geometrically overconstrained, so another source of** this information is needed. During one point contact, the values of $\Delta F_x/\Delta Z$ and $\Delta F_\theta/\Delta Z$ will have the same sign. At the onset of two point contact, the value of $\Delta F_\theta/\Delta Z$ will reverse in sign, because the contact forces are now reducing the misalignment of the peg rather than increasing it. The value of $\Delta F_\theta/\Delta Z$ is a monotonically decreasing function of the relative Z position of the peg and hole and is therefore a source of the necessary relative position information. Use of this variable assumes that K_x, K_θ, and distances to the center of compliance are set so that the peg will not break two point contact due to motion along the assembly path [Whitney 82] and that the tolerances of the parts being assembled are small compared to the clearance between the parts. Methods that could potentially reduce these constraints are discussed in the Chapter 6.

- During two point contact, the angle of contact is constrained by the geometry of the parts, so $\Delta F_x/\Delta Z$ is no longer a valid source of angular misalignment information, but knowledge of the relative Z position of the parts and knowledge of the existence of two point contact is sufficient to determine the relative angular orientation of the peg and the hole. This method of determining the misalignment works if the assumptions mentioned above are valid.

- The relative position of the parts in X is determined as in the case of one point contact.

In an idealized assembly cell with no position errors, measurement of the Z position will produce redundant information. In a real system, this information can be extremely useful. For example, if a part slips in the robot gripper or is presented incorrectly to the robot, the inconsistency in the measured Z position and the value of F_θ/F_x may be a convenient way of determining that there is something wrong with the system. For small contact angles, the force measurements in Z are too corrupted by friction while the peg is contacting the side of the hole to be of any use as state information (in a frictionless system, F_z can be used to determine the relative orientation of the parts in both one and two point contact), but the Z force information is useful for determining contact between the peg and the bottom of the hole.

2.3 Representation of Information

The approach presented in this thesis uses a discrete form of information storage. I began this work based on the assumption that people (who are remarkably good at assembly) were probably not resolving forces to more than 5 or 10 levels on any particular assembly task. It seemed reasonable that a robot assembly cell should be able to get by with a similar number. Figure 2.2 shows three examples of state measurements mapped into six discrete state values. The range of the states was determined experimentally. The range of the state variables was estimated based on the incremental move size along the NAP and the system stiffnesses. Several assemblies were performed, and then the ranges were adjusted so that the state measurements spanned the range of each of the states. I used the same procedure for the simulations and the real assembly tests.

As coarse as the discretization in Figure 2.2 is, it still generates a huge number of state combinations. Fortunately, the fact that there are 6^7 *mathematically* possible system states for a system with 7 states discretized into 6 regions is of no particular relevance to the real assembly task because most of these states are geometrically unrealizible. Experiments verified simulation results that indicated that no more than 300 out of the possible 279,936 states would ever be visited.

Computer memory and data access times are not the most severe constraint

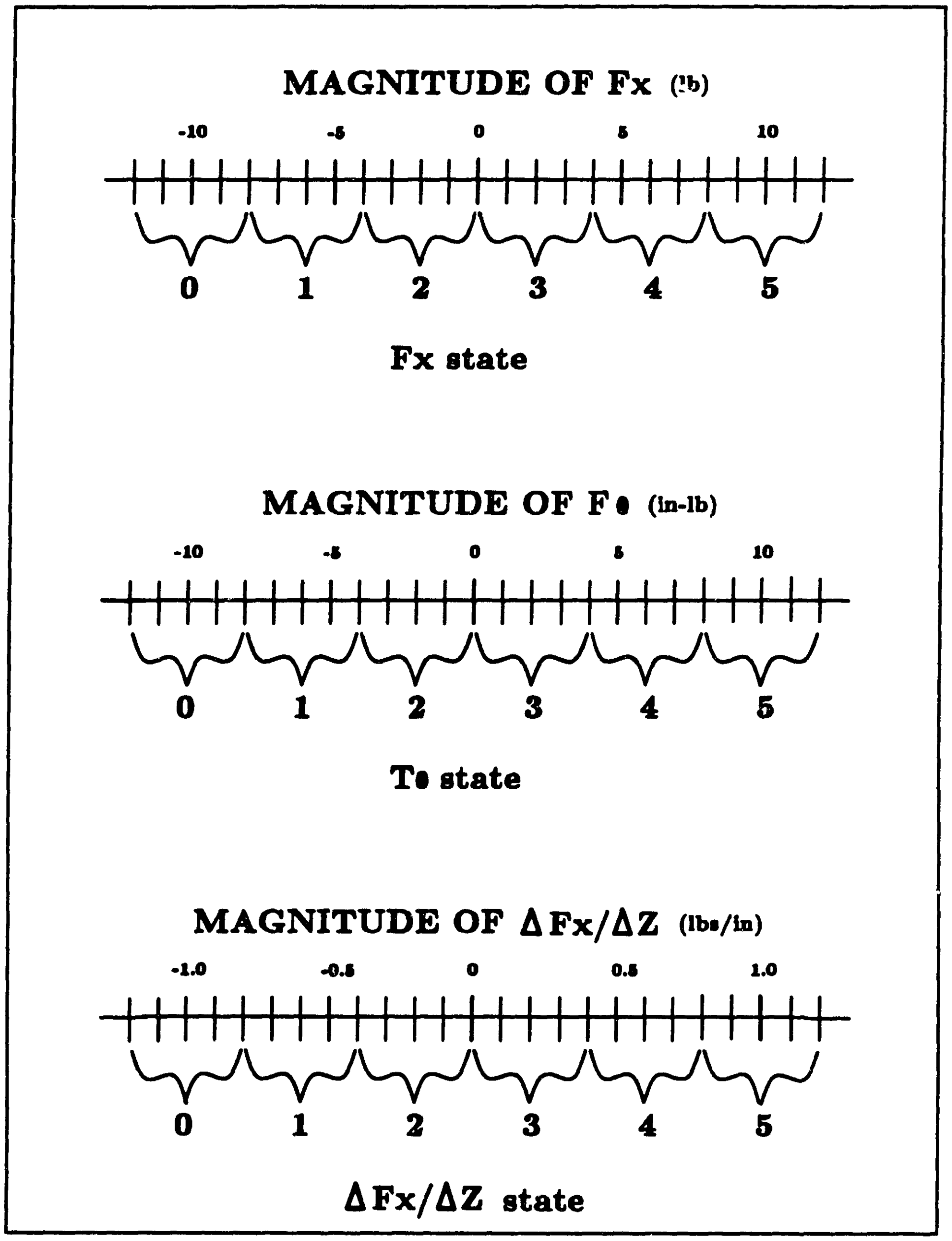

Figure 2.2: Mapping of State Measurements Into State Values - The range of the state measurements is normalized by mapping onto the range of the state values (in this case 0 through 5). Analog force measurements are mapped into one of the 6 discrete regions.

on the convergence rate of the learning system. *The time required to visit all of the states during the learning phase of the algorithm is by far the most time consuming part of the learning process.* In general, it will be desirable to make a compromise between maximizing information content with high state resolution and the time required to visit all of those states during the learning phase of the algorithm.

In the absence of friction and robot errors, the maximum discretization level is set by the noise and drift in the force sensor. In addition to the forces generated by the contact angle between the parts, friction generates an additional force of up to $\pm\mu N$. For measurements of states where the angle of contact is small relative to the incremental move direction, e. g. forces in the Z direction in a peg and hole assembly with small orientation errors, the friction forces are typically the most significant measurement error in the states. In a well behaved system (good backdriveability, high position resolution, etc.), if the friction is extremely low or if the angle of contact is large (approaching 90 degrees), the optimum discretization of the states based on sensor and robot errors may be so fine that the total number of states is unmanageably large. For example, a system using 7 states with a discretization level of 100 has 100^7 *mathematically* possible states. Fortunately, there is rarely a need for this level of resolution.

The angular resolution required to avoid jamming sets the lower bound on the required resolution of the force derivative states. This angle is a strong function of the clearance ratio of the parts. Also of importance are the coefficient of friction between the parts and the compliance of the parts and robot system. For example, the 2.60 inch peg and 2.70 inch hole that I used in most of my tests required a resolution of about 2 degrees. Uniform discretization and an expected maximum error of ±5 degrees then requires 5 states. The 1.000 inch steel peg and 1.010 inch aluminum hole that I used in some of my tests required a resolution of about

0.25 degrees. Given the same initial errors as the previous system, ass many as 40 states would be required to avoid jamming, depending on the incremental move size and the assembly strategy used. A direct approach to the problem of having too many states is to use variable state resolution and range [Simons, et. al. 85], e.g. in the example presented above, there will clearly be no need to maintain force derivative states near the bottom of the hole that correspond to angular errors of 5 degrees when the geometry of the peg and hole has constrained the angular error to less than .1 degrees. An indirect approach is to avoid using discrete states at all by making some assumptions about the nature of the data and then, based on a small subset of the states, generate a *functional representation* of the data that spans the space. Drawbacks of this approach include the time required to recompute the function to include new data and the difficulty of preserving the temporal relationship of the data [Connell and Utgoff 87].

2.4 The Assembly Algorithm

2.4.1 The Assembly Process

A flow chart of the assembly process in a logic branching form is shown in Figure 2.3. In block 1, the system is initialized with the peg above the hole and concentric with it. A random error in Θ was then introduced by tilting the peg (Figure 2.4).

I used a Gaussian distribution with the tails clipped at 2σ to generate random initial position errors. These values represent the cumulative effect of peg, hole, and robot position errors. A clipped Gaussian distribution was easy to generate and I felt that it was a reasonable approximation to the errors encountered when peg-like parts are stored in loose fitting pallet. Motion of the peg along the

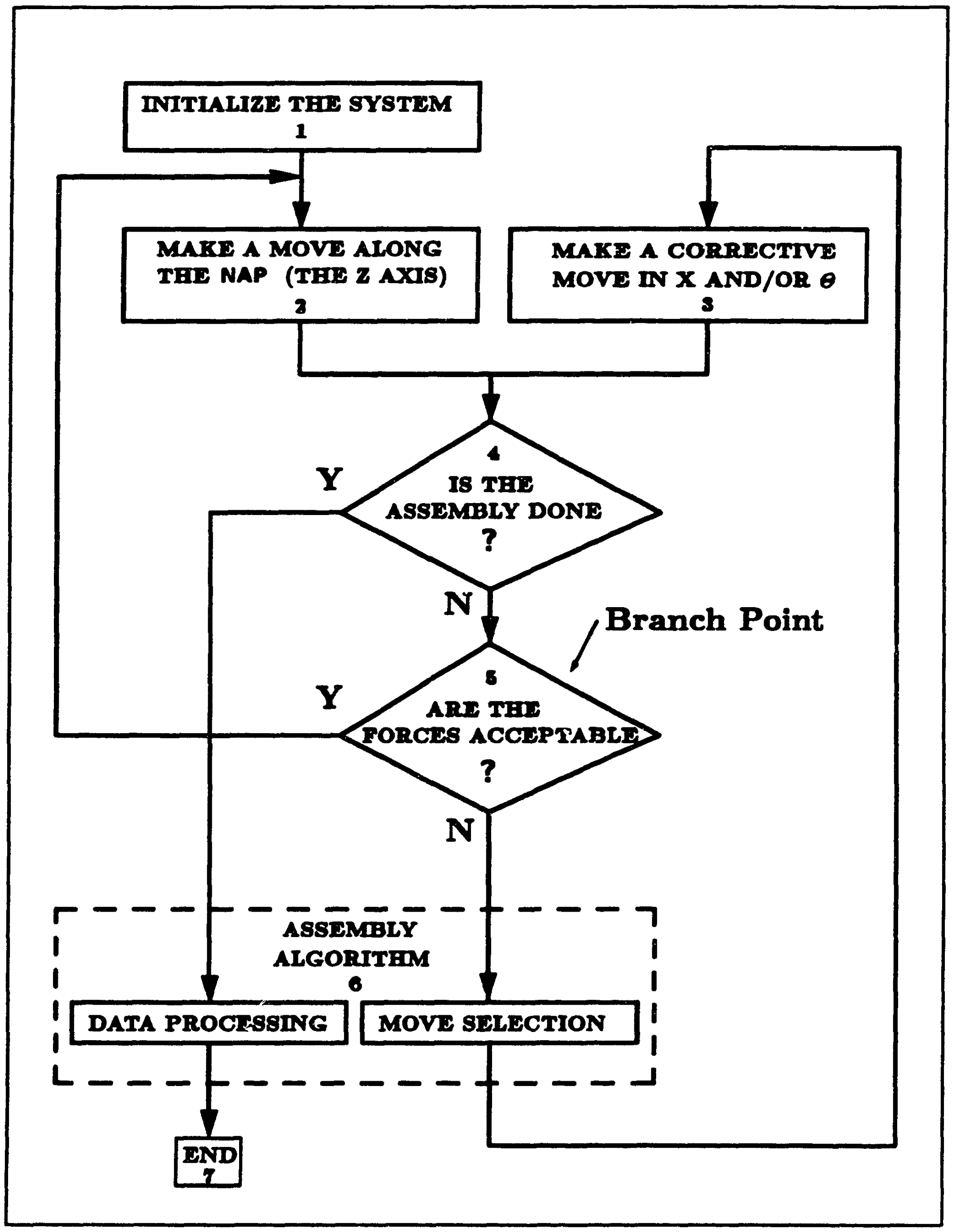

Figure 2.3: Flow Chart of the Logic-branching Assembly Algorithm

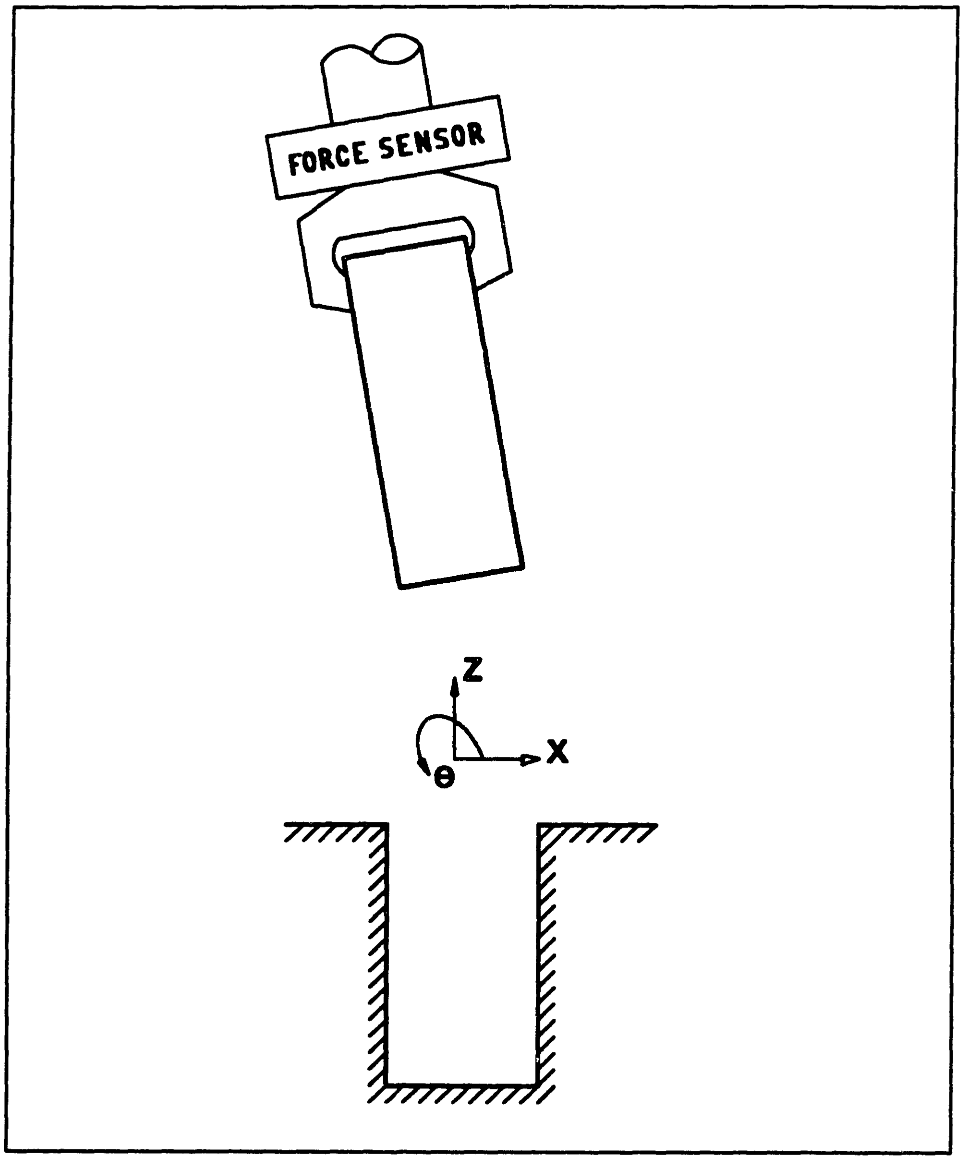

Figure 2.4: Initial Position of The Peg and Hole

nominal assembly path occurs at block 2 in Figure 2.3. The peg is moved along the assembly path in small increments. The increments vary from 0.005 to 0.100 inch, depending on the stiffness of the system. Typically, the increments were chosen so that the forces in X and Θ would never be more than 1 lb or 1 inch-lb above the maximum allowed by the force limit function defined below. Contact forces are calculated after each move. Incremental motion continues along the NAP until the force limit branch point is reached or exceeded. The following equation is an example of a simple force limit function used to determine a branch point:

$$\sqrt{F_x^2 + F_z^2 + (1inch \times F_\theta)^2} \geq \text{FORCE-LIMIT} \tag{2.1}$$

When this limit is violated, the program moves to block 6. The assembly algorithm is called upon to generate a response to the force limit that was exceeded. The program then returns to block 2 to continue incrementing along the NAP (the Z axis in the peg and hole case). The program continues in this manner until the assembly fails or the termination conditions are met. Failure results when the entire tree in Figure 2.6 is searched without getting under the FORCE-LIMIT. These termination conditions are: the Z position of the peg being within the maximum Z error of the bottom of the hole; and the force in Z being 5 times the maximum value that could be generated during 1 or 2 point contact of the peg on the sides of the hole. This is a simple heuristic, but in general, a successful one. When necessary, a reasonable extension to this condition is to include the rate of change of the force in Z ($\Delta F_z/\Delta Z$). This aids in the differentiation of contact with the bottom of the hole and conditions such as jamming and wedging. Work in the Design For Assembly (DFA) field addresses this and related issues in automated assembly [Boothroyd 1980].

2.4.2 The Assembly Algorithm

The assembly algorithm does the following things:

- When an assembly is complete, calculate the distance in X and Θ from the state associated with each branch point to the final position of the peg.
- Add this information to a list of previously visited states.
- If any of these states are visited during future assemblies, use the stored information to determine the next move of the robot.

When the contact forces exceed the force limit (block 5 in Figure 2.3), the assembly algorithm is called upon to select a direction (or direction and distance) in which to move. At the beginning of an assembly run, the list of previously visited states is empty, so the algorithm initially selects corrective moves at random. As the algorithm learns (more assemblies are performed), the list of visited states and the amount of information associated with each of those states grows. Eventually all of the states that can be visited will have been visited, and the output of the algorithm is always based on information from previous assembly trials. A few elements from the list are shown in Figure 2.5. I decided to keep track of only the moves in X because I found the results were clearer and easier to present using only one variable. X and Θ are geometrically coupled, so, barring any pathological algorithm behavior (none was observed in the simulation or on the real system), tracking the performance of the algorithm only with respect to X was sufficient. The first direction in which to move in Θ was always chosen randomly.

The output of the algorithm, generally a move of set length in X, typically 0.010 inch or 0.001 radian, is passed to block 3, the section of the program that deals with corrective moves. After the move is made, the contact forces are checked.

```
3 2 5 0 5 -8 -9 -8 -12 0 0 0 0 0 0 4
2 4 0 5 2 9 10 8 14 4 6 8 6 5 4 101
2 3 0 5 2 10 7 11 10 9 8 7 5 3 9 42
3 2 5 0 4 -16 -7 -12 -11 -9 -8 -11 -10 -18 -24 11
3 1 5 0 3 -18 -16 -16 -15 -16 -14 -12 -10 -19 -17 108
3 2 5 0 3 -20 -17 -20 -17 -16 -17 -18 -17 -22 -21 36
2 4 0 5 1 3 6 4 3 5 5 8 6 4 4 53
1 4 0 5 1 6 5 10 7 8 7 6 3 0 0 8
1 5 0 5 2 11 9 11 13 15 14 12 10 13 12 11
3 2 3 2 0 -2 1 1 1 1 1 3 2 1 -2 34
3 2 3 2 3 -7 -9 -7 -12 -9 0 0 0 0 0 5
3 2 5 1 0 -1 -2 -1 -1 -1 -1 -1 -1 -1 -1 19
3 2 4 1 3 -4 -4 -7 -6 -8 -7 -10 -8 -4 -15 15
```

Figure 2.5: Data Stored by the Assembly Algorithm. The first five elements of each row are the values of the states F_x, F_θ, $\Delta F_x/\Delta Z$, $\Delta F_\theta/\Delta Z$, and Z. The next 10 elements are the distances (in units of incremental moves in X) from that state to the position of the peg at the completion of the assembly. The last element of each row is the number of times that state has been visited since during a series of assemblies.

If the forces have gone up, the move is undone and a move in another direction is tried. If the forces go down, the program continues to make moves until the forces are again below the force limit. For simplicity, I chose to restrict the first corrective move to be a move in X. When I did this, the choice of X was an arbitrary decision because only one corrective move was allowed in X and Θ. I soon realized that this approach would work only for a small range of hole aspect ratios and centers of rotation (for Θ). Given a system outside of that range, an incremental move size in Θ could not be chosen that would work for all of the peg and hole orientations that could be encountered during an assembly. I then modified the algorithm so multiple moves in X and Θ were allowed. Moves in X are made until either the forces go below the force limit or the forces begin to go up again. If the forces are still over the limit, moves in Θ are made until the forces go under the limit. If necessary, the program makes a depth first search through all four combinations of corrective moves. The search tree in X and Θ is shown in Figure 2.6. On the hardware I used, always moving in X first is very desirable because the robot could make much finer moves in Θ than in X, i.e. the largest translation in X of the peg or hole due to the smallest possible move in Θ was considerably smaller than the smallest move in X. This meant that if there was a value of Θ that would bring the forces under the FORCE-LIMIT for the given position of the X axis, the discrete moves in Θ were less likely to jump over the value.

Unlike absolute position information, incremental position information of high accuracy is often easily accessible. This information is typically a full order of magnitude better than absolute position information [Gordon, et al., 1983]. A robot that has a global position accuracy of ± 0.010 inch will typically have a ± 0.001 inch accuracy for an incremental move of 0.100 inch, a distance sufficient to correct for part position errors in a typical assembly cell. The algorithm keeps

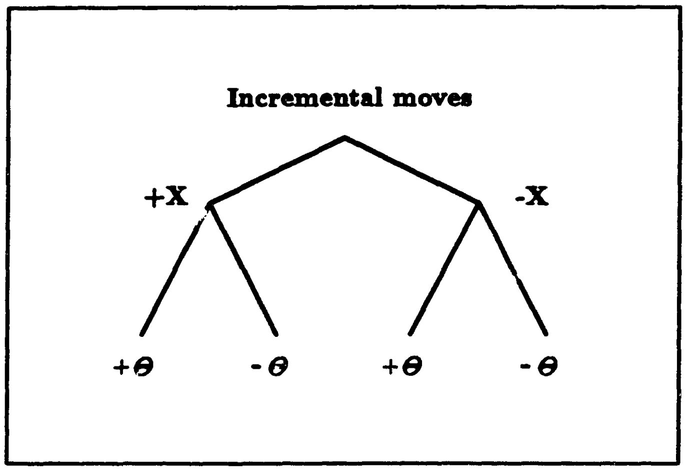

Figure 2.6: The Search Tree for the Corrective Moves in X and Θ

track of the robot position that corresponds to each of the states from which corrective moves are made during an assembly. Once the assembly is complete, the relative position of the parts is known at least to within the tolerance and clearance of the parts. *The algorithm then backs up through the data generated during the assembly and determines the location of each visited state relative to the final position and consequently to the NAP. This transforms the local information collected during the assembly into global information, a major improvement in the information content of the data.* Without the information gained by backtracking, the algorithm is just as likely to reduce the contact forces during one point contact by rotating out of the hole (increasing the angular misalignment) as it is to rotate into the hole. This behavior will continue until two point contact is established. From that point on, there is only one move direction that will reduce the forces. Without backtracking, the algorithm can learn the correct response during two point contact, but can not learn to avoid globally unproductive moves during one point contact because the algorithm's measure of move quality is local, i.e. the move is defined to be good if the forces go down, not because the move necessarily benefitted the assembly. Any move that rotates the peg out of the hole must later be corrected by making a move in the opposite direction. This shortcoming could potentially be dealt with by using a model of the geometries of the peg and hole, but avoiding the use of system models is one of the main goals of this thesis. This problem is solved, without the use of a model, by the backtracking. If the algorithm is set up properly, subsequent assemblies that encounter previously visited states will consistently make moves from these states toward the NAP. The algorithm makes only enough moves (of set incremental length) to reduce the forces in X and Θ to satisfy the FORCE-LIMIT constraint. The relative position information makes the transformation from force sensor information to relative part position direct and complete and has no reliance on any modeling of the peg, hole, and robot system. All that needs to be known is the nominal assembly path and a termination condition.

Chapter 3

Test Hardware

3.1 The Robot/Controller System

3.1.1 The Robot

The MIT Precision Assembly Robot (MITPAR) was used for the work in this thesis. It was designed as a test bed for performing assembly tasks and as a vehicle for studying robot control strategies. A detailed discussion of the design of the robot is in Appendix A. The robot was designed with a rather unorthodox geometry (Figure 1.2). Two axes of the wrist (Pan and Tilt) are attached to the base rather than to the arm. The configuration of these two axes is similar to that of a standard two axis welding table. Most six degree-of-freedom robots are designed with their axes in series. Because the weight of the wrist axes must be carried by the other axes, the wrist axes are usually much less stiff than the axes further back in the kinematic chain. By mounting the wrist to the base, we were able to make the two wrist axes very stiff without compromising the performance of the other axes.

3.1.2 The Computer/Interface System

Three separate microprocessors run simultaneously and divide the work of controlling the robot. They share a common backplane which has a complement of interface cards to connect to the outside world. A Sun 3/180 Unix workstation provides a development environment and data storage. The interface cards are : a digital to analog converter board, an analog to digital converter board, five optical encoder reading cards, a digital I/O board, and extra memory. The backplanes of the Sun and the VMEbus expansion box are connected together so that data can be transferred to and from the Unix system. More detailed specifications are listed in Appendix B.

The Condor system was used for the programming/operating interface to the computer. "Condor" refers to the computational architecture and programming environment developed at the MIT Artificial Intelligence Laboratory by [Narasimhan, et. al. 88]. The Condor system is composed of two parts. The first part is a collection of subroutines and libraries that handle all communication between programs and the I/O boards. These libraries also provide standard methods of inter-processor communication and timed interrupt routines for servo loops. The second part of Condor is a user interface between the Sun computer and the Ironics processor boards. The interface, called "Xcondor", runs under the X11 windowing system. It opens a window which is connected to a process on the Sun system and windows which are pseudo-terminals that connect to each of the processor boards on the VMEbus system. It also provides fast downloading over the extended VMEbus. The user writes separate programs for each Ironics processor, compiles them on the Sun, runs the Xcondor program, downloads the programs over the VMEbus to the processor boards, and starts them running. The user communicates with each program through the pseudo-terminal interface.

3.1.3 The Force Sensor

The test setup used a JR3 6-axis force sensor (JR3, Inc. Woodland, CA) We chose this sensor for it's flexible interface. The interfaces offered are: raw analog strain gage signals (the axis decoupling matrix is supplied in the documentation), RS232, digitized parallel I/O, and decoupled analog. The particular model that we have is rated at ± 25 lbs in X and Y, ± 50 lbs in Z and ± 50 in-lbs in Mx, My, and Mz. We used the parallel digital port to transmit the force data from the JR3 interface board to the Condor system because it was convenient. The higher data acquisition rate achievable through use of the analog strain gage signals and A/D converters on the CONDOR system was not required for the assembly algorithms. The JR3 controller board uses 15 bit A/D converters. The first 2 bits were noisy using the default filter cutoff frequency of 163 Hz. We did not lower the filter cutoff frequency because we did not need the resolution and other people in the lab were doing closed loop force control work which required the high bandwidth. Because the logic branching approach used for this work does not use closed loop force feedback, the approach is very tolerant of non-ideal sensor behavior. Typical force state resolutions for the assembly algorithms are around 0.5 lbs. Friction between the parts generates typical variations in force readings of around 10% of the measured forces, so the effects of sensor resolution (0.0015 lb) and sensor noise (± 0.006) lb are completely swamped out by the measurement errors due to the frictional forces. Sensor drift is not a problem because the assembly is performed quickly relative to the time constant of the drift. A system initialization routine resets the force sensor offsets before each assembly.

3.1.4 Controller Design

A detailed discussion of the design and implementation of the servo/controller system can be found in the Appendix A. I was able to get adequate performance from the robot by using PD position controllers with digital state estimators for velocity. Settling time could be reduced significantly by using a trajectory controller or input filtering, but I did not feel that the potential increase in speed would be worth the effort and increased system complexity for the experiments I was running. Using velocity control is one of several issues discussed in the Future Work section.

Friction prevents the robot from settling at a commanded setpoint. This is a serious problem for an assembly strategy that attempts to maintain or reach a certain force. Fortunately, logic branching only requires that some change of state occur - usually movement of one or more of the axes. The practical restriction that the use of a PD controller placed on the assembly algorithms was a minimum move size of 0.002 inch. A position command smaller than this would not necessarily produce any motion at all because the preloaded bearings in the MITPAR joints generate a considerable amount of stiction and coulomb friction.

3.1.5 Transition from Simulation to Reality

The longest and most complicated functions in the simulation were the ones that modeled the geometry and interaction of the peg and hole. None of these were needed for the real system. The functions that remained were relatively simple, so I decided to rewrite them in C rather than deal with the possible mysteries generated by the use of the C-to-LISP interface that would have been required. The interface would have been necessary because the Condor robot control software development system is written in C.

I made several additions to the assembly algorithm to accommodate the behavior of the real robot system. Preliminary trials on the real system were run with a two second pause between the moves to allow the system to settle. The settling times of the different axes vary considerably, so I replaced the open loop pause with a function that monitors the force sensor and sets a flag when the system (forces) has settled. In addition to speeding up the assembly process, this function effectively decouples the system dynamics from the assembly algorithm. Changes in the robot dynamics (structural or controller) show up only as differences in assembly times.

The simulations used the conventional peg, hole, and force sensor arrangement shown in Figure 2.4. The peculiar geometry of the MITPAR encouraged the use of the peg and hole configurations shown in Figure 3.1.

The state information was originally stored in array form. The array spanned the entire space of each of the states. Data stored in this form was both quick to access by the computer and easy to decipher by the debugger (me). The data was stored in a list for the real assembly tests because there is a very limited amount of memory on the system ($\approx 500k$). Access time on the real system was not an issue. For example, one set of assemblies was performed with 7 states (Fx, Fz, Fθ, ΔFx/ΔZ, ΔFz/ΔZ, ΔTt/Δz, Z) at a discretization level of 6. This results in 279,936 possible combinations of states. Most of these states will never be encountered because they are physically impossible, e.g. Fx and Fθ will always have the same sign during one point contact. Most of the remaining states will not be encountered because no corrective moves are required when the system exists in them, e.g. if all of the states have values near zero. During more than 200 assembly trials, only around 300 of the 279,936 states are visited.

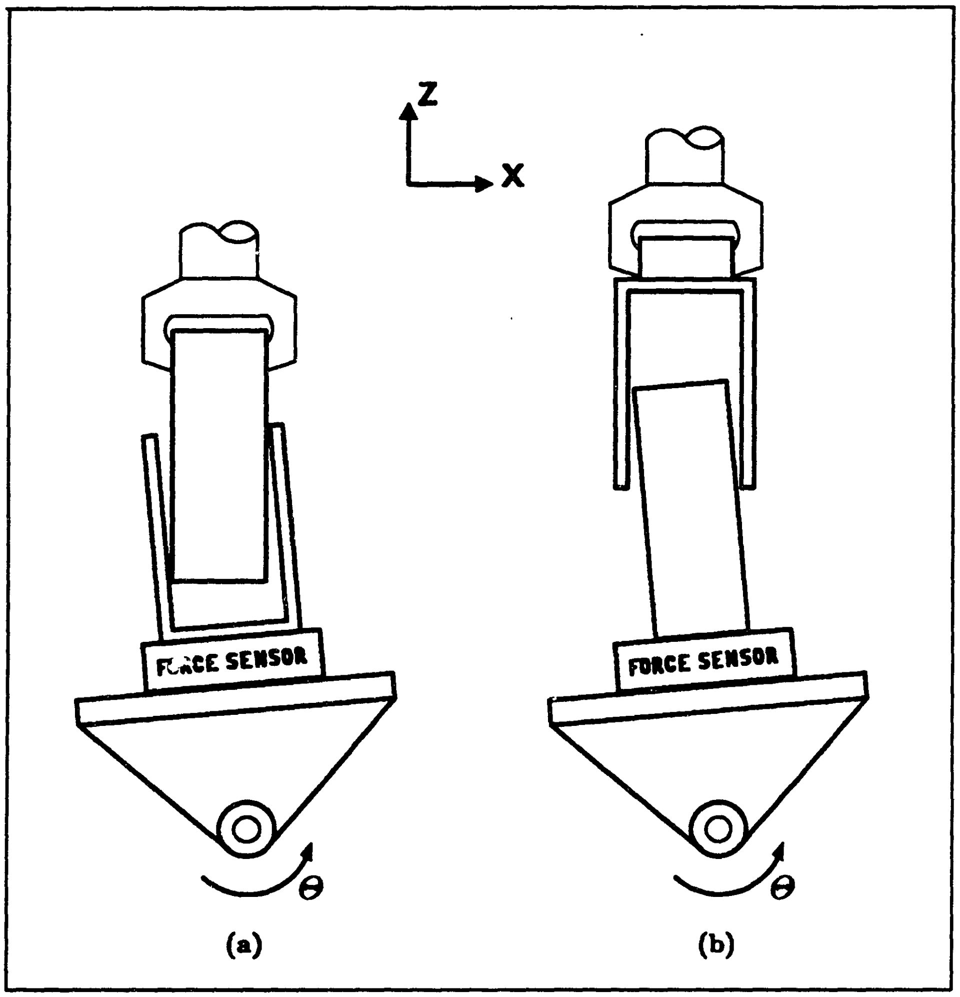

Figure 3.1: Peg, Hole, and Force Sensor Arrangement for the Hardware Tests

Over 90% of the states are actually visited for the first time during the first 20 assemblies. Checking *all* of the these 300 states to see if they match the present state of the system is in requires only 0.003 seconds.

3.2 The Pegs and Holes

Coke cans were used for the peg in most of the trials. The cans offer an excellent balance of uniformity, availability, stiffness, and crushability. When there is a problem with the controller, the cans crush without damaging the force sensor. U-shaped pieces of aluminum were used for the holes. The sides of the hole were bolted to the base. The bolt holes were slotted so the clearance between the peg and the hole could be adjusted. Clearance ratios of 0.02 to 0.05 were used. Lower clearance ratios caused the ratio of contact forces to sensed forces to go up to the point where the can could be deformed significantly before the forces measured by the force sensor were large enough to be of use. The last set of assembly trials was performed with a 1 inch diameter steel peg. The mating holes were machined from 2 inch diameter aluminum bar stock. The clearance ratios varied from 0.001 to 0.005.

Chapter 4

Experimental Results

4.1 State Resolution

I ran a series of tests to determine how repeatable the force and torque measurements were. The results of the tests indicate where the upper bound on the resolution of the states should be set. The tests were run by tilting the hole and then bringing the peg into contact by incrementing along the NAP (in 0.025 inch increments) until the force limit (FORCE_LIMIT) was exceeded. The forces, torques, and force and torque derivatives were measured/calculated, and then the peg was withdrawn. This process was repeated 10 times with the hole at 1 to 5 degrees, in increments of 0.5 degrees. Figure 4.1 shows the results of these tests for 1 degree (top) and 1.5 degrees (bottom).

Of particular relevance to the feasibility of the basic approach I am taking is the very clear and consistent difference in the $\Delta F_\theta/\Delta Z$ values resulting from a difference of 0.5 degrees in contact angle. Also of interest are the errors present in the data. Encoder resolution combined with a 0.003 inch random position error in Z is responsible for the bimodal distribution of the data for the 1 degree case. One

Fx	Fz	Ft	DFx/DZ	DFz/DZ	DFt/DZ
-0.634995	-0.002670	2.185816	7.170104	0.094605	-24.517784
-0.444451	-0.001907	1.543727	5.984502	0.265503	-20.617599
-0.444870	0.003509	1.542659	6.364447	-0.079346	-21.661301
-0.630913	-0.004044	2.177729	7.499693	0.021362	-25.439409
-0.441818	-0.007095	1.564021	5.972293	0.314331	-21.496500
-0.434303	-0.001297	1.536098	5.876162	0.271607	-20.989914
-0.636482	-0.006332	2.193445	7.647706	0.112915	-25.750689
-0.447541	-0.001450	1.546626	6.150823	0.186157	-20.690842
-0.636788	-0.004349	2.192835	7.516478	-0.064087	-25.561480
-0.634499	-0.013886	2.211756	7.591248	0.354004	-25.640825

Fx	Fz	Ft	DFx/DZ	DFz/DZ	DFt/DZ
-0.859185	-0.021744	2.946463	10.308849	0.195313	-36.132660
-0.850945	-0.035629	2.939444	10.572824	0.747681	-37.920986
-0.842553	-0.032578	2.900535	10.061650	0.851441	-36.077732
-0.837212	-0.031891	2.890616	10.661318	0.778199	-38.024754
-0.827294	-0.028534	2.852469	10.026547	0.711060	-35.583347
-0.821495	-0.027618	2.832023	10.023496	0.430298	-35.748154
-0.815125	-0.048142	2.835074	10.157773	1.290894	-36.724705
-0.809975	-0.045700	2.817680	10.124204	1.327516	-36.547707
-0.801926	-0.046387	2.794182	10.014340	1.190186	-35.687141
-0.792999	-0.060883	2.780296	9.974668	1.486208	-36.163216

Figure 4.1: State Values For Contact at 1 Degree (top) and 1.5 Degrees (bottom)

encoder count in θ corresponds to about 0.0001 radians. The deflection in Θ due to a 0.100 inch move in Z along a surface inclined at 1 degree to the direction of motion is about 0.0004 radians. The small error in Z results in deflections in the Θ direction of either three or four counts. The PD controller is used to position the axis has a gain of about 0.5 in-lb per count.

The drift in the F_x data for the 1.5 degree tests is due to repeatability errors in the bearings in the robot. The bearings that the Z axis runs on return to almost, but not quite, the same position during the tests. Runout in these bearings produces repeatability errors in the XY plane (see Appendix A for robot construction details). No evidence of this appears in the derivative states. Their values, as you would expect, are essentially independent of slight variations in the X position of the peg.

4.2 Performance of the Assembly Algorithm

The system was set up with the 2.5 inch peg (coke can) using a clearance ratio of 0.04 and random errors in initial orientation of up to $\pm$ 5 degrees. No force state/corrective move information was collected from the first thirty assemblies in order to establish a baseline system performance level. Force state/corrective move information was collected and used during all the successive trials until 150 assemblies were completed.

Equation 4.1 is used to show how well the system is performing. Equation 4.2 shows how much learned information was available to the assembly algorithm. All of the zero-move assemblies (assemblies with initial errors so small that no corrective moves were necessary) were extracted from the assembly data. I did this because I felt that the convergence behavior of the algorithm should not be influenced by random assemblies that did not offer an opportunity to learn. I experimented with simply plotting the algorithm behavior against the running total of the number of states visited. This, I felt, was a reasonable way to represent the data because the number of visited states is a more fundamental parameter with respect to the learning algorithm than is the number of assemblies. Unfortunately, most people (myself included) are so used to thinking in terms of learning vs. number of assemblies attempted, that this form of display met with some resistance. I decided to keep track of only the moves in X because I found the results were clearer and easier to present using only one variable. X and Θ are geometrically coupled, so, barring any pathological algorithm behavior (none was observed in the simulation or on the real system), tracking the performance of the algorithm only with respect to X was sufficient.

Some filtering was required to make the trends stand out clearly. Each plotted point is the result of applying Equation 4.1 and Equation 4.2 to only the next 20 assemblies, e.g. the data point for assembly number 17 is the result of applying the equations to assemblies 17 through 36.

I had some concerns about the repeatability of the learning and convergence rate of the system, so I ran several independent series of assemblies using the basic assembly algorithm. Figure 4.2 shows reasonably consistent behavior for four runs of 150 assemblies.

$$\mathrm{PI1}_n = \frac{\sum_{i=n}^{n+20} minimum\ required\ number\ of\ moves}{\sum_{i=n}^{n+20} number\ of\ moves\ made} \tag{4.1}$$

$$\mathrm{PI2}_n = \frac{\sum_{i=n}^{n+20} number\ of\ new\ states\ visited}{\sum_{i=n}^{n+20} total\ number\ of\ states\ visited} \tag{4.2}$$

Applying Equations 4.1 & 4.1 to the data collected during real assembly trials are shown in Figure 4.3. Figure 4.4 is the result of plotting the output of

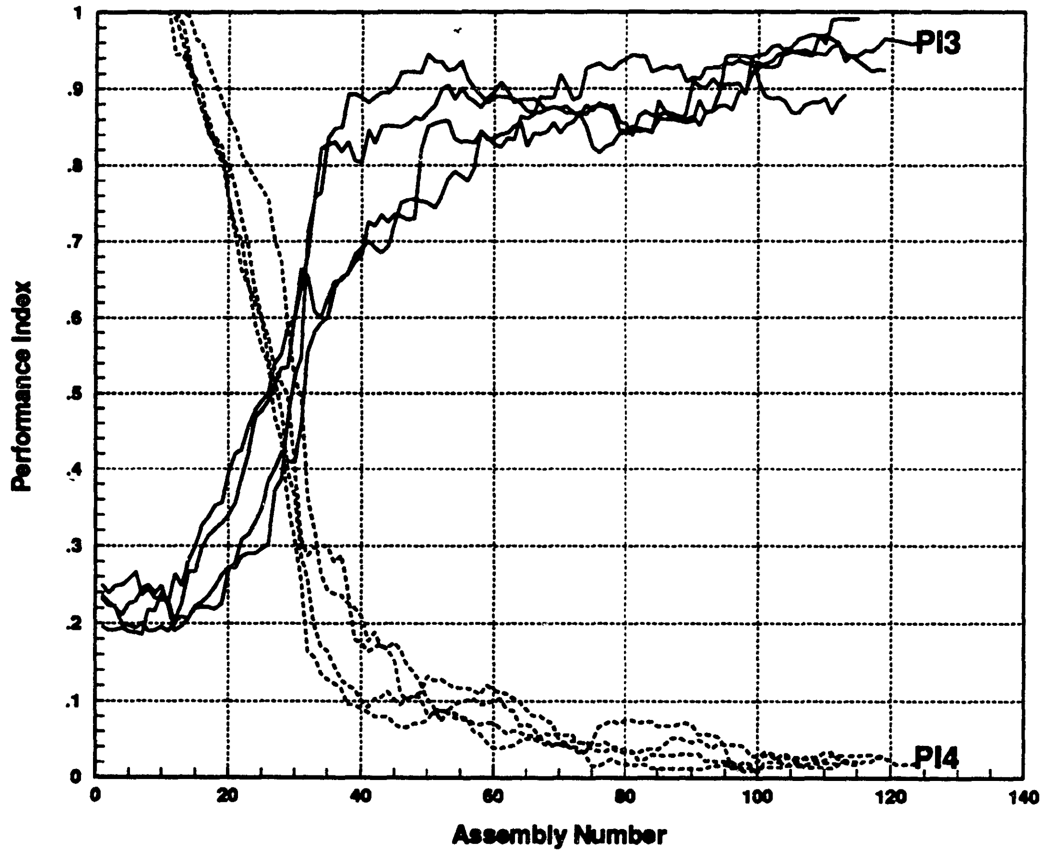

Figure 4.2: Repeatability of the Convergence Rate of the Assembly Algorithm

Equations 4.1 & 4.2 against each other. The plot gives a good indication of the nondimensionalized *learning behavior* of the learning algorithm. The roughly straight, diagonal line indicates that the algorithm performs roughly in proportion to the amount of information available to it. Algorithms that begin to perform much better with only a little knowledge will produce traces curved to the upper right. Algorithms that require almost complete knowledge before they are successful produce traces curved to the lower left. Every tenth data point is plotted with a star to show the learning behavior relative to the number of assemblies. The "scribble" at the lower right indicates convergence.

4.3 Evaluation of Modifications to Assembly Algorithm

4.3.1 The Bold Move Strategy - Test10

Because the force/move pairs have global significance, there is an opportunity to switch from a learning mode to a "bold move" mode after some learning has taken place. In this mode the robot brings the peg and hole into contact, looks up the move data associated with the state the peg and hole are in, and then makes all of the corrective moves suggested by the data at once. The performance of this strategy is somewhat at odds with the desire for quick convergence of the learning curve in that using the bold move strategy ideally results in only one state being visited during each assembly.

The distribution of move distances in a given state gets tighter as the discretization level of the states gets higher until the randomness in the force measurements due to friction and system errors become a significant part of the state size. This point was reached typically around a state discretization level of 5 to 10, giving

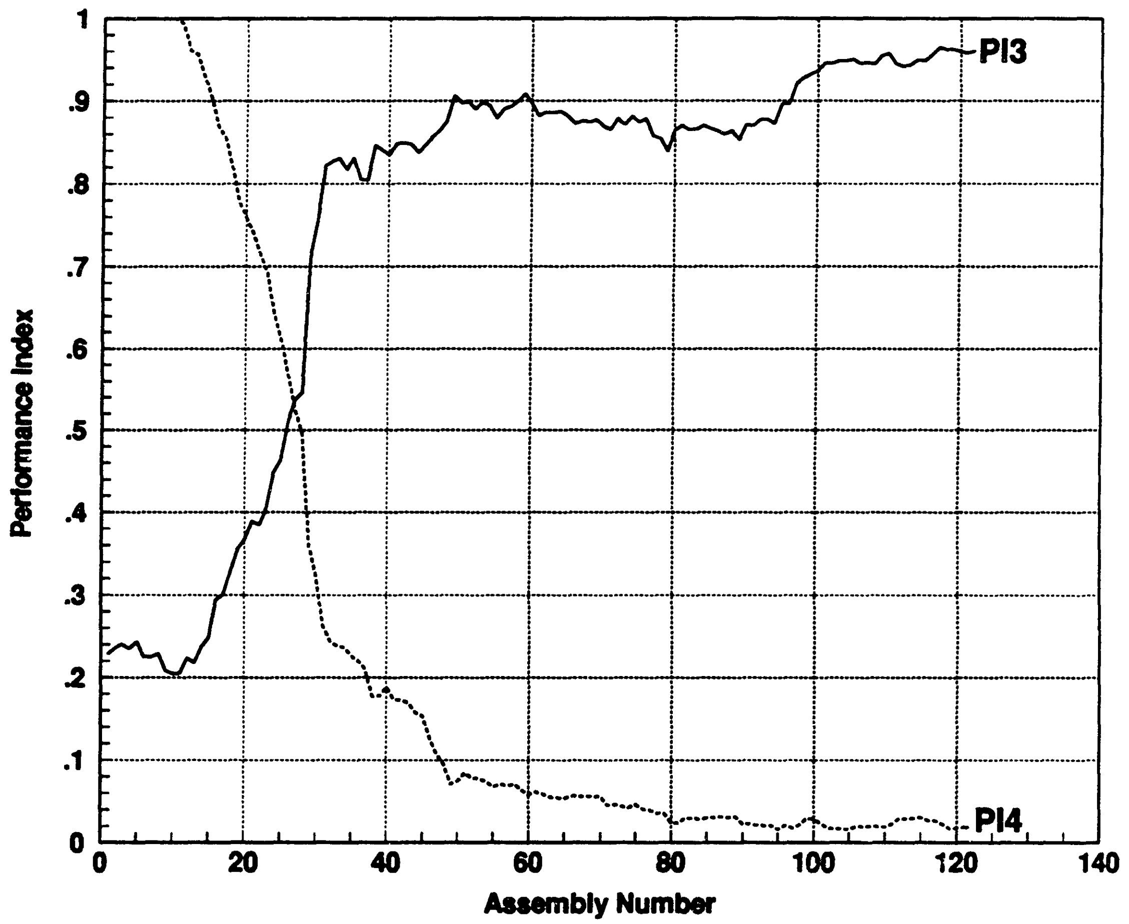

Figure 4.3: Convergence of the Assembly Algorithm

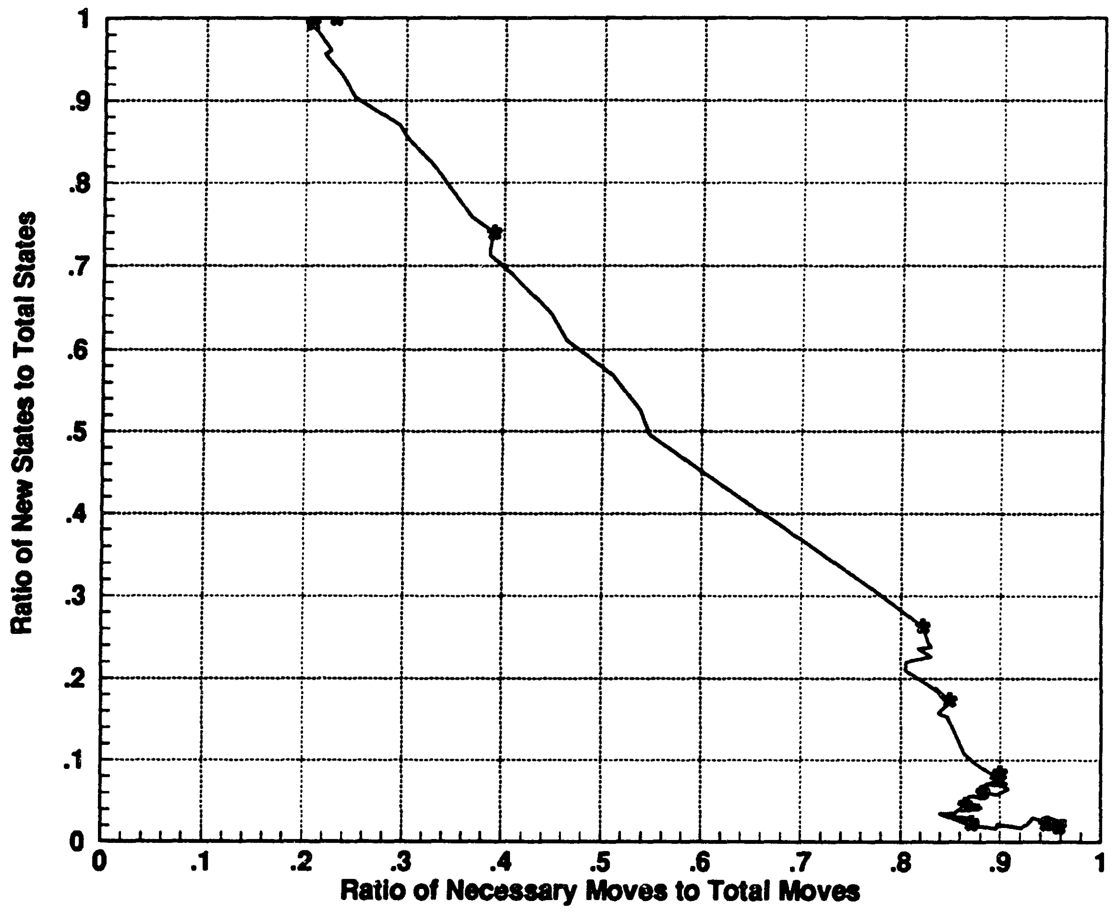

Figure 4.4: Learning Rate of the Assembly Algorithm

an orientation accuracy of 1 degree in a system with initial errors of ± 5 degrees.

Supervised learning using initial contact information combined with a bold move response can often result in an effective, but fragile assembly algorithm [Dufay and Latombe 84]. The possible number of initial contact states is small, so learning (or teaching) is fast, but there is enough uncertainty in the force information that the likelihood of missing the goal state and encountering new states is high. When the bold move strategy of Test10 was run using data collected by the series of runs that was used to produce Figure 4.7, the performance was impressive. The assembly often required only one branch point. The branching decision commanded all of the necessary corrections in X and Θ, and continued incrementing along the NAP until the peg reached the bottom of the hole. The raw assembly data shows this behavior clearly (Figure 4.5). In assembly 2, the branch point occurred at a state that contained move data that was 2 moves short of being enough to allow the peg to reach the bottom of the hole without exceeding the force limit and generating more branch points. The 2 remaining branch points were made at states that had been visited before. The correct move in X and Θ was made in both cases. In assembly 4, the bold move was also too short. This time a new state was visited when the force limit was exceeded. The wrong corrective move in X was tried first and then reversed, which accounts for the number of total X moves being larger by 2 than the number of necessary X moves.

4.3.2 Flexible Range of States - Test3

To get an idea what price I was paying for using such a crude state discretization scheme, I introduced some flexibility into the range of the states. The basic assembly algorithm has a maximum and minimum value for each of the states

assembly number	1	2	3	4	5	6	7
necessary X moves	18	11	5	18	8	4	12
total X moves	18	11	5	20	8	4	12
new states visited	0	0	0	1	0	0	0
branch points	1	3	1	2	1	1	1

Figure 4.5: Assembly Data for Test10

and a set discretization level within those limits. Any state measurement outside of the specified range was given a state value corresponding to the nearest end of the set range. This gave me control over the maximum number of states, but reduced the accuracy of the states at the ends of the range. State measurements occasionally extended beyond the set range because I was trying to achieve a compromise between corrupting the states at the ends of the range (with too small a range) and loosing information by not spanning the range (with too large a range). For Test3, I dropped the constraint on the number of discrete regions in each state and simply set the range and the number of discrete states that spanned the range. Now, for example, if the range of Fx had been set at $\pm 1lb$, and the discretization level set at 10 (states 0 to 9), a state measurement of Fx of 1.1 lbs was now assigned a state value of 10. More states were then visited during the 100 assembly trials (252 vs. 103), resulting in a slower convergence rate (Figure 4.7). On the plus side, the distribution of distances to the NAP, was, as expected, tighter than when using the basic algorithm. The bold move strategy of Test10 would benefit from this situation.

The sample states from Test2 and Test3 (Figure 4.6) show the different distributions clearly. The two states from Test3 would be considered one state in Test2.

	state	X distances to the goal state										visits
Test2	3 2 5 0 2	-5	-12	-3	-5	-3	-7	-6	-16	-14	-7	45
Test3	3 2 5 0 2	-4	-9	-6	-6	-5	-6	-5	-4	-6	-5	19
	3 2 7 -2 2	-13	-11	-9	-12	-11	-13	-11	-9	-11	-11	37

Figure 4.6: Sample States from Test2 and Test3

4.3.3 Force Limits With Hysteresis - Test4

As in the basic assembly algorithm, Test4 was set up so that exceeding the force limit triggered a corrective move. In Test4, the corrective moves were continued until a lower force limit threshold was reached. If the lower limit could not be reached, the system did the best it could and continued the assembly. The force limits were set at 2 and 0.5. A setting much lower than 0.5 tended to get lost in the noise. The only first order effect of introducing hysteresis into the force limit was the slower convergence rate relative to the number of assemblies performed. This was due to the smaller number of states visited during each assembly. A possible, but unverified second order effect is that the larger number of moves encouraged in the hysteresis mode might reduce the likelihood of the peg jamming or wedging in the hole on at least some of the assemblies. Wedging would then be avoided during any future assemblies with similar errors in initial position.

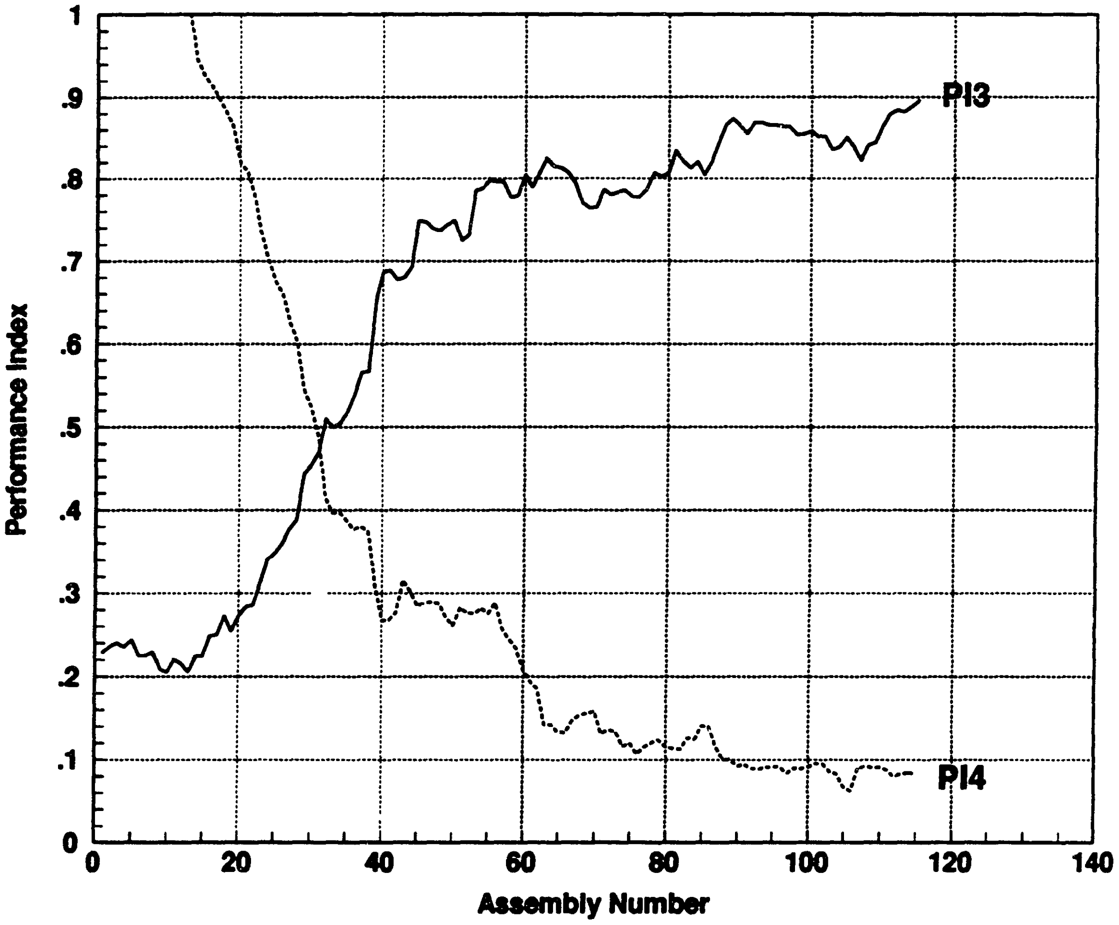

Figure 4.7: Convergence Using a Flexible Range of States

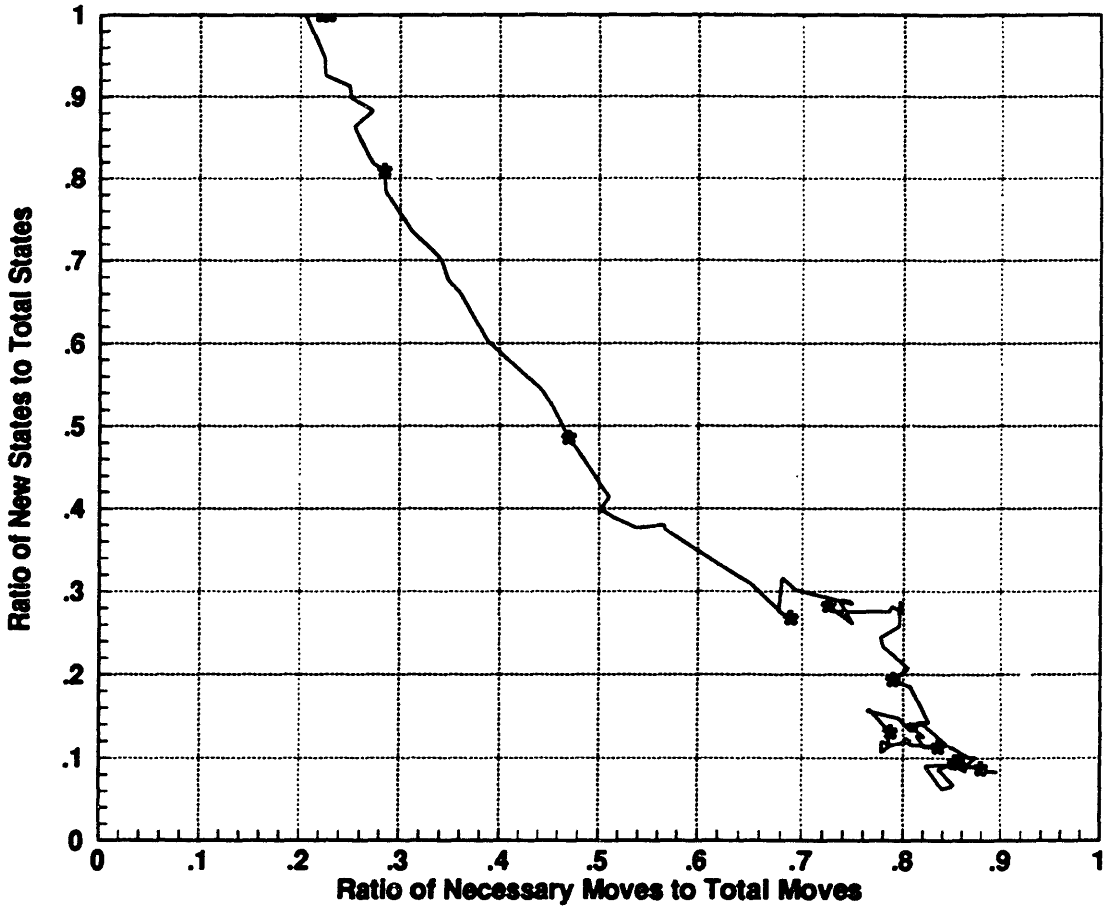

Figure 4.8: Learning Using a Flexible Range of States

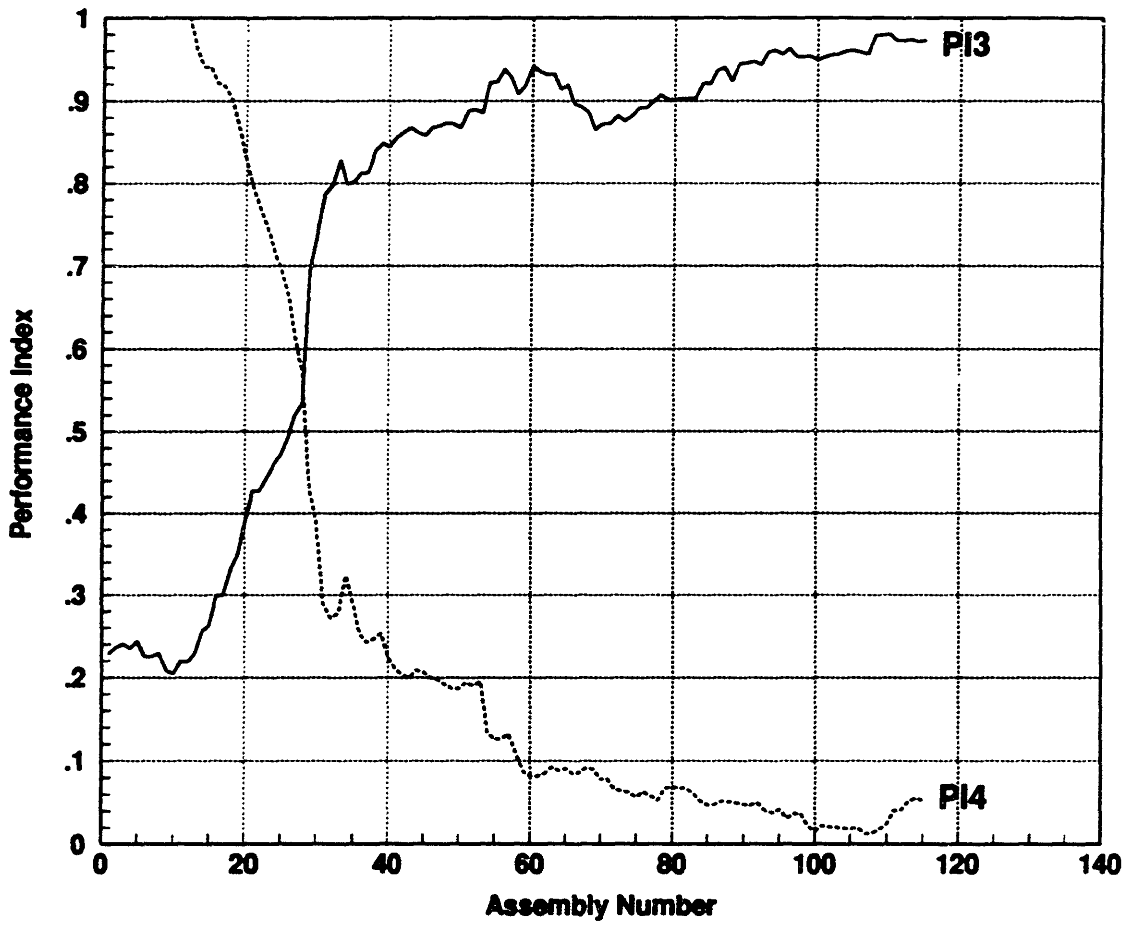

Figure 4.9: Convergence Using Hysteresis in the Force Limit

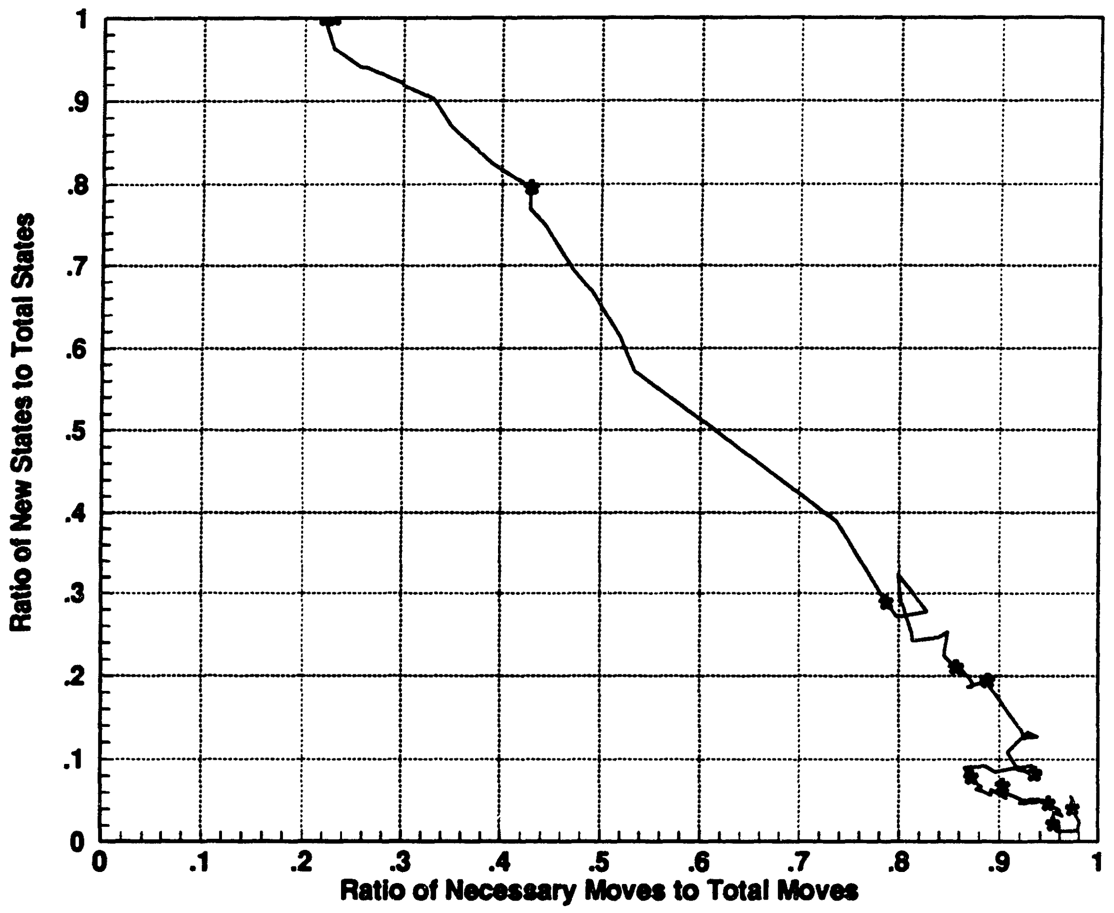

Figure 4.10: Learning Using Hysteresis in the Force Limit

4.3.4 Reduced State Vector - Force States and the Z Position State - Test5

I ran a series of tests using only Fx, Ft, and Z with the expectation that the results would verify the need for keeping track of the force derivative states as well as the force states. It was during these test that I became aware of the effect of having an initial error in X as well as in Θ. If there is no error in X, the bottom of the peg always starts out over the bottom of the hole, and there is a set relationship between the initial error in Θ and the values of Fx, Ft, and Z. The behavior of Test5 was actually quite good when there was no X error. After correcting my oversight (adding an initial position error in X) I made another series of tests. The results are shown in Figure 4.12 and Figure 4.13. The choice of system parameters, in particular, the discretization level of the states, caused the two configurations shown (exaggerated) in Figure 4.11 to be interpreted as the same state for a small range of initial errors in X and Θ. A higher discretization level could patch this problem but not completely solve it, because the discretization level would get unmanageably large as the peg and hole clearance ratios got smaller. In addition to the convergence being lower than when using the force derivative states (Figure 4.3), the distribution of move distances for each state is large. This problem is not correctable. It is a systematic shortcoming of Test5 due to the lack of force derivatives or other source of relative angle information. The bold move strategy can not work well with this reduced set of states.

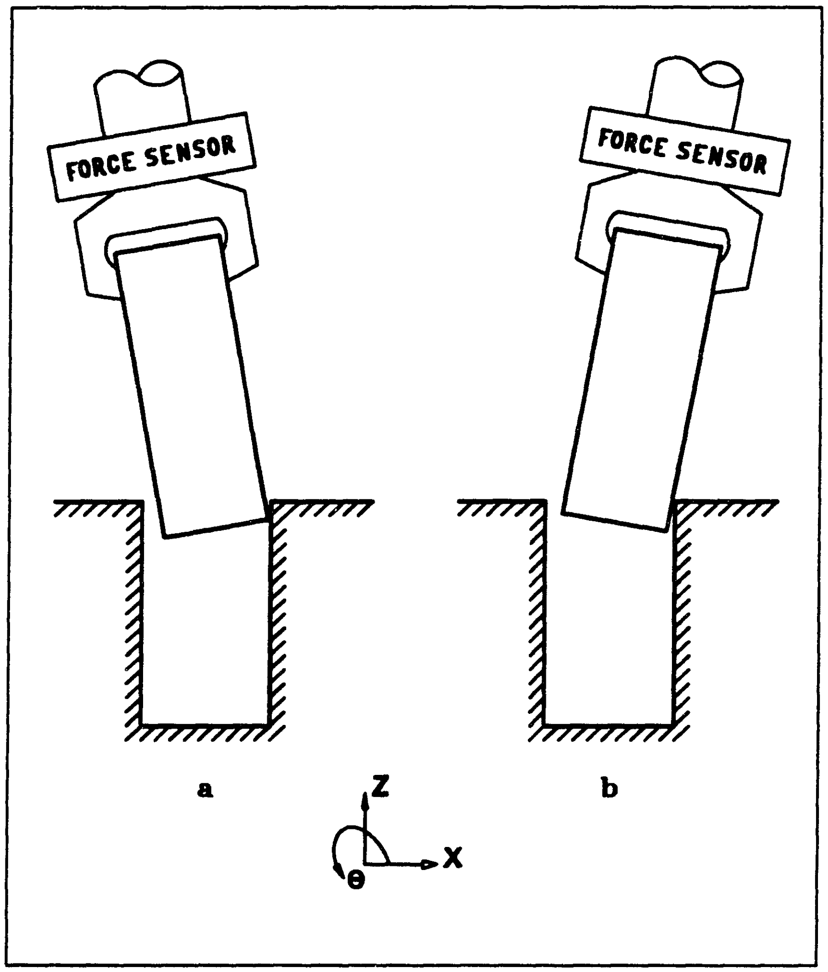

Figure 4.11: Two Identical States for Test5

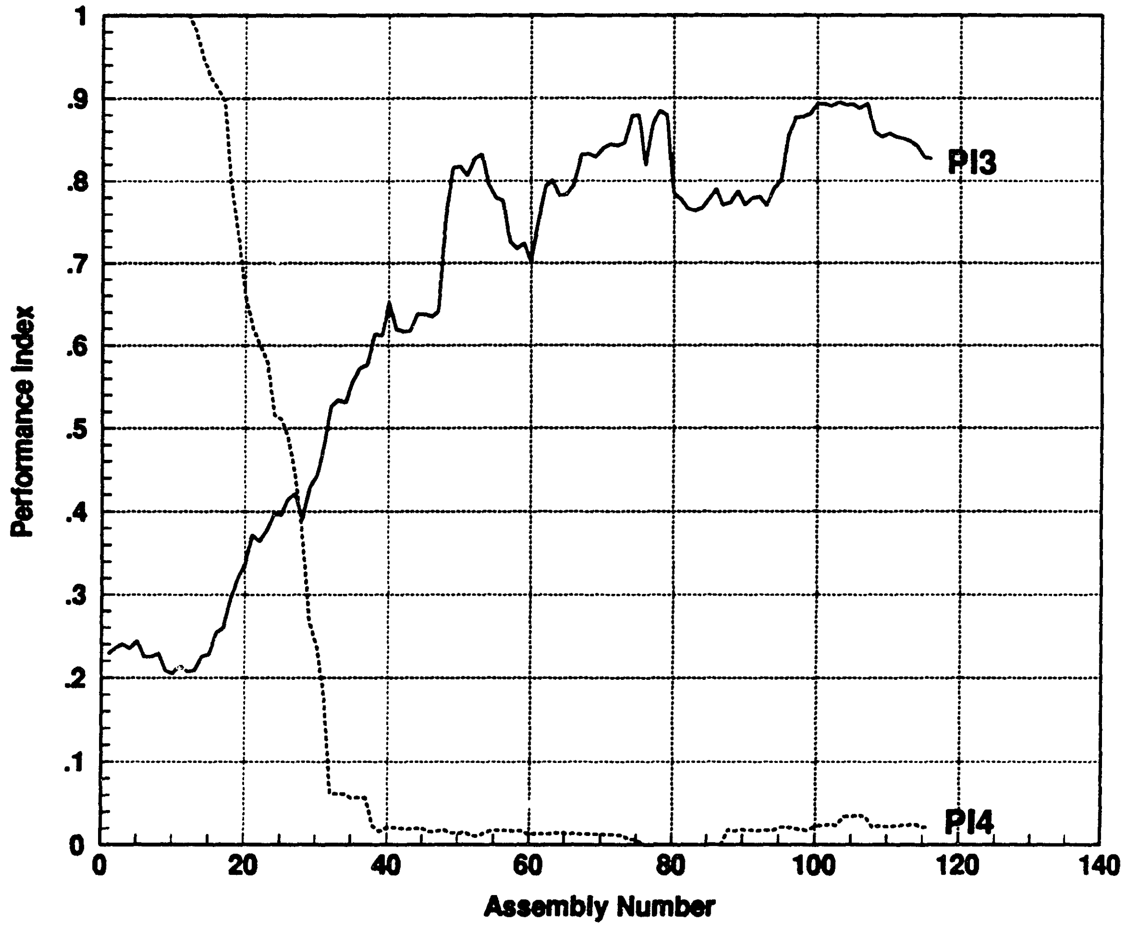

Figure 4.12: Convergence Using Only the Force States and the Z Position State

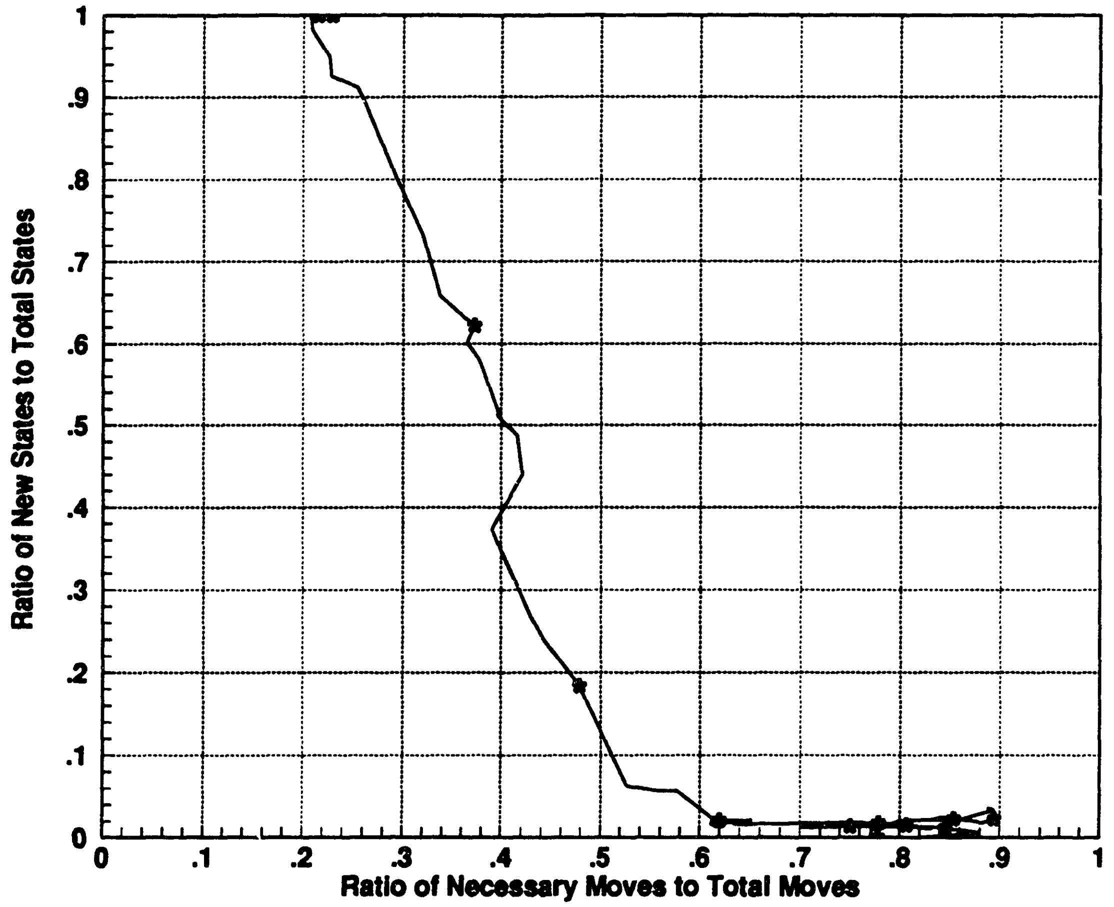

Figure 4.13: Learning Using Only the Force States and the Z Position State

4.3.5 Reduced State Vector - Force Derivative States and the Z Position State - Test6

Are the force derivative states and the Z position enough? The results of Test6 (Figure 4.14) suggest that they are. I believe that the force states should still be used because they are a direct measure of the relative position of the parts along the axis of the NAP. Simply measuring the Z axis position is an indirect measurement that is sensitive to unanticipated part position errors.

4.3.6 Passive Wrist Compliance - Test7

I ran a series of tests with a compliant element (a Lord #J4624-32 sandwich mount vibration isolator) mounted between the Z axis and the peg. Before starting the tests I had assumed that the added compliance would help by allowing larger moves to be made. The larger moves would be less effected by the nonideal positioning behavior of the robot. The addition of the compliant element changed the stiffness of the peg from 600 lbs/inch and 300 inch-lbs/degree to 5 lbs/inch and 1 inch-lb/degree measured at the bottom of the peg. The results in Figures 4.16 & 4.17 show that, at least for the compliances I chose, my intuition was very wrong. The basic assembly algorithm will behave well with a wide range of compliances, but it is sensitive to the locations of the centers of those compliances. This sensitivity is due to the uniform range of the states, i.e. the maximum and minimum values and the discretization are constant over the NAP. In order to get a force measurement during one point contact that is above the noise it was necessary to set the incremental move length in Z to 0.200 inch. Following my normal procedure, I adjusted the range of the states so the state measurements spanned the state space. The system behaved well until the peg came into two point contact with the hole. The force and torque measurements jumped by an

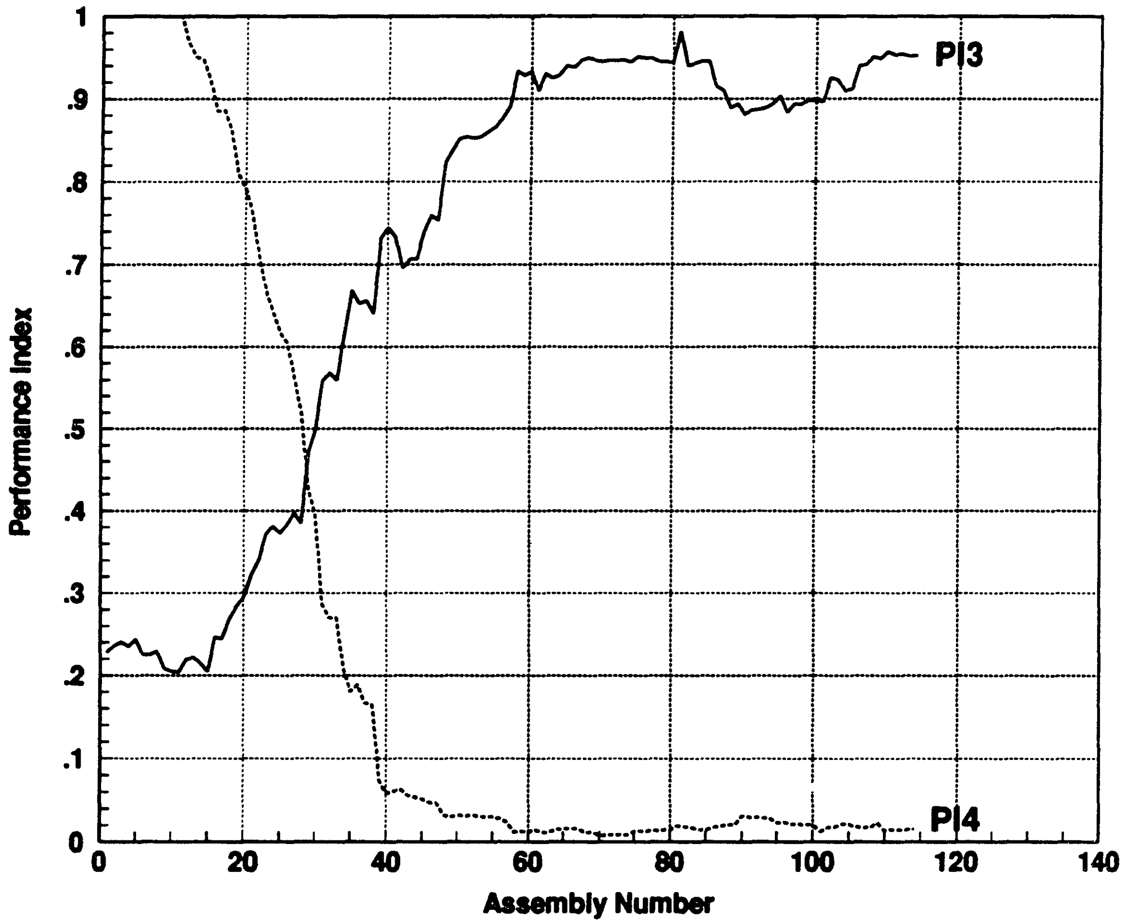

Figure 4.14: Convergence Using Only the Force Derivative States and the Z Position State

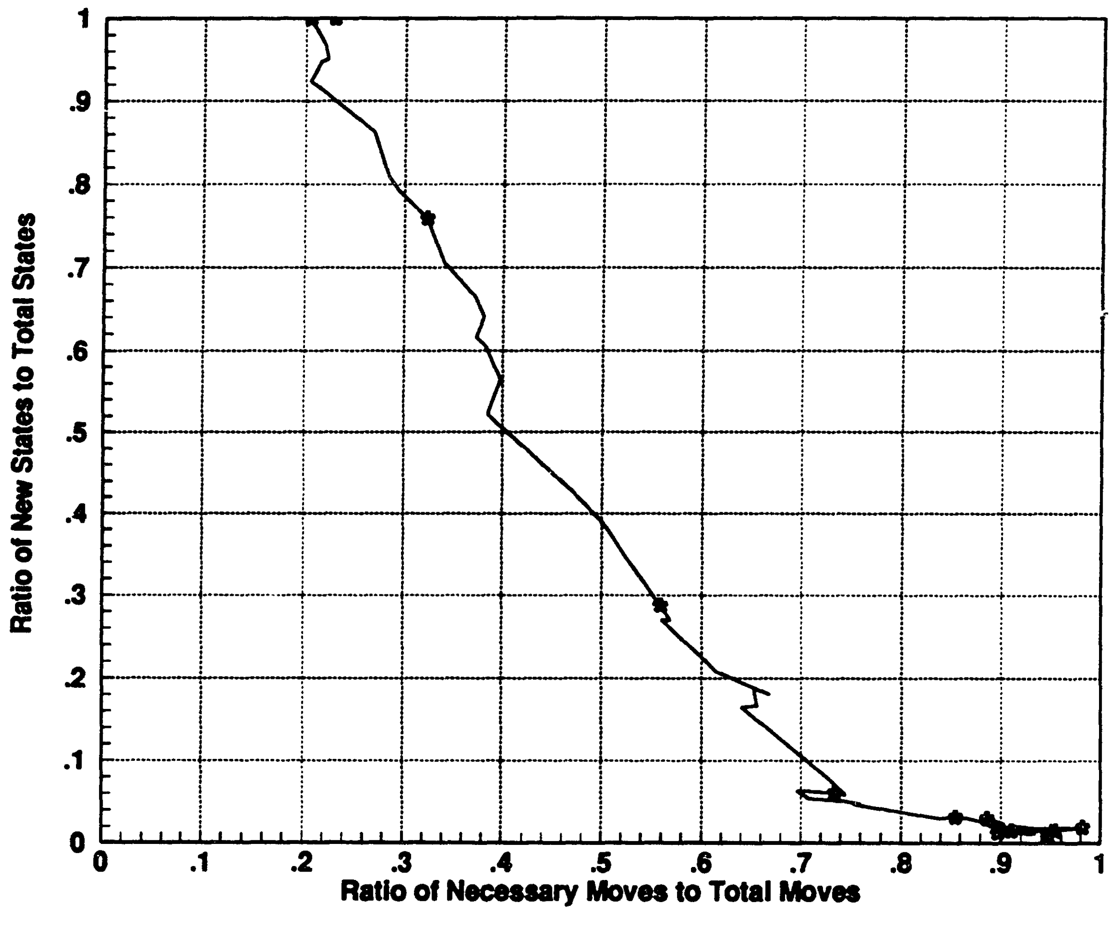

Figure 4.15: Learning Using Only the Force Derivative States and the Z Position State

order of magnitude. This was due to the effective X stiffness of the peg rising almost to the value measured before insertion of the compliant element. The added compliance was not the primary problem, the *abrupt change of compliance* was. Within certain limits, a more complex form of the learning algorithm could deal with the behavior of this system by having state ranges that varied with position along the NAP. The performance of the algorithm gradually declines as the rotational stiffness goes down no matter what is done about the range of the states because the algorithm uses the position of the peg when the assembly is complete to calculate the relative move lengths from each of the states visited during the assembly. If the peg has no rotational stiffness, there is no information available to determine the orientation of the peg.

4.3.7 Low State Discretization - Test9

The upper limit of state resolution is set by the effect of friction and other system errors at around 10 (see the Section 4.1). The lower limit is 2. I gave this state resolution a try out of curiosity. The assembly algorithm behavior using a state discretization of 2 is shown in Figures 4.19 & 4.20. The behavior was quite interesting. The system would often begin to converge quickly but then begin to make incorrect moves. This behavior was initiated when a corrective move was made during 2 point contact (Figure 4.18a) that resulted in one point contact (Figure 4.18b).

Usually the peg would then be incremented down until contact with the other side of the hole was reestablished or until the assembly was complete. However, if the forces were just below the force limit and there was a small orientation error in the hole, incrementing in Z could cause the force limit to be exceeded during single point contact. Then, because the discretization level was set at 2, the system was

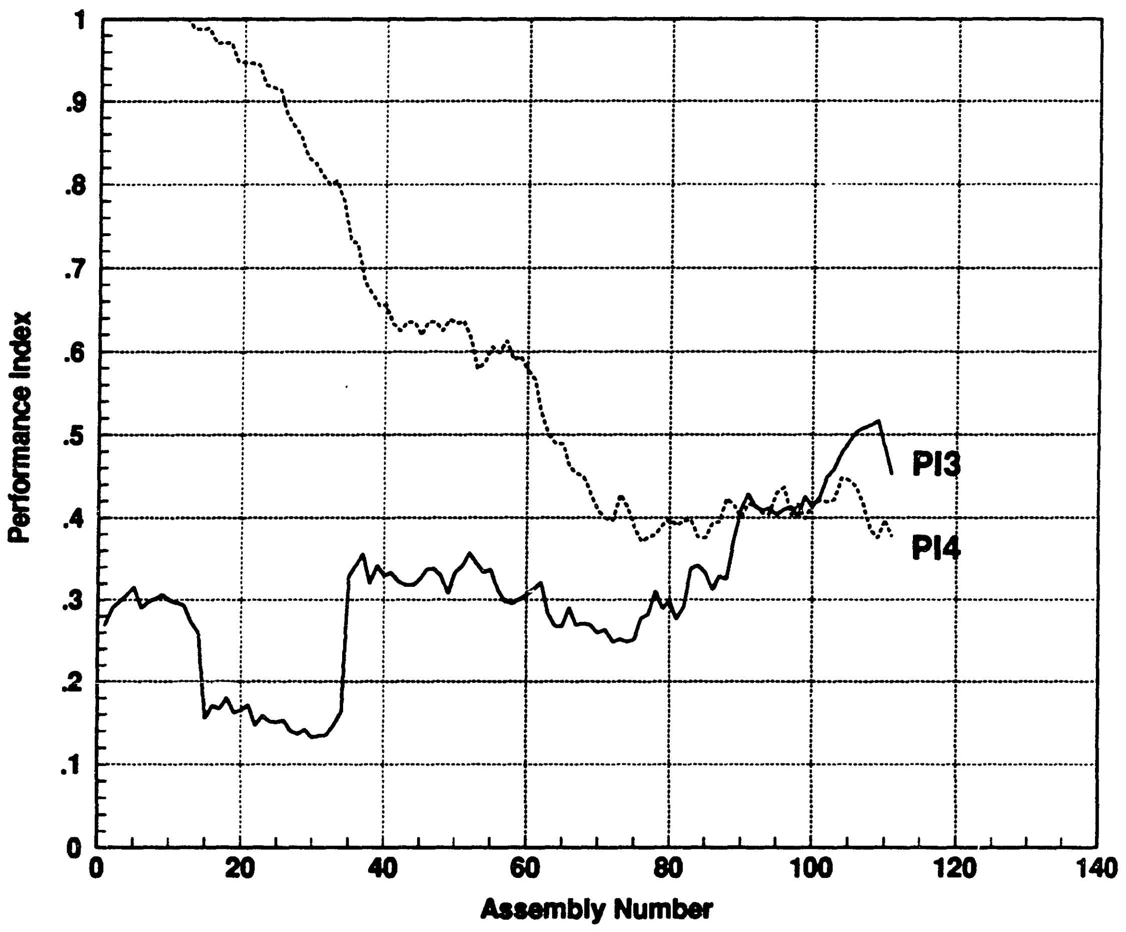

Figure 4.16: Convergence Using Passive Wrist Compliance

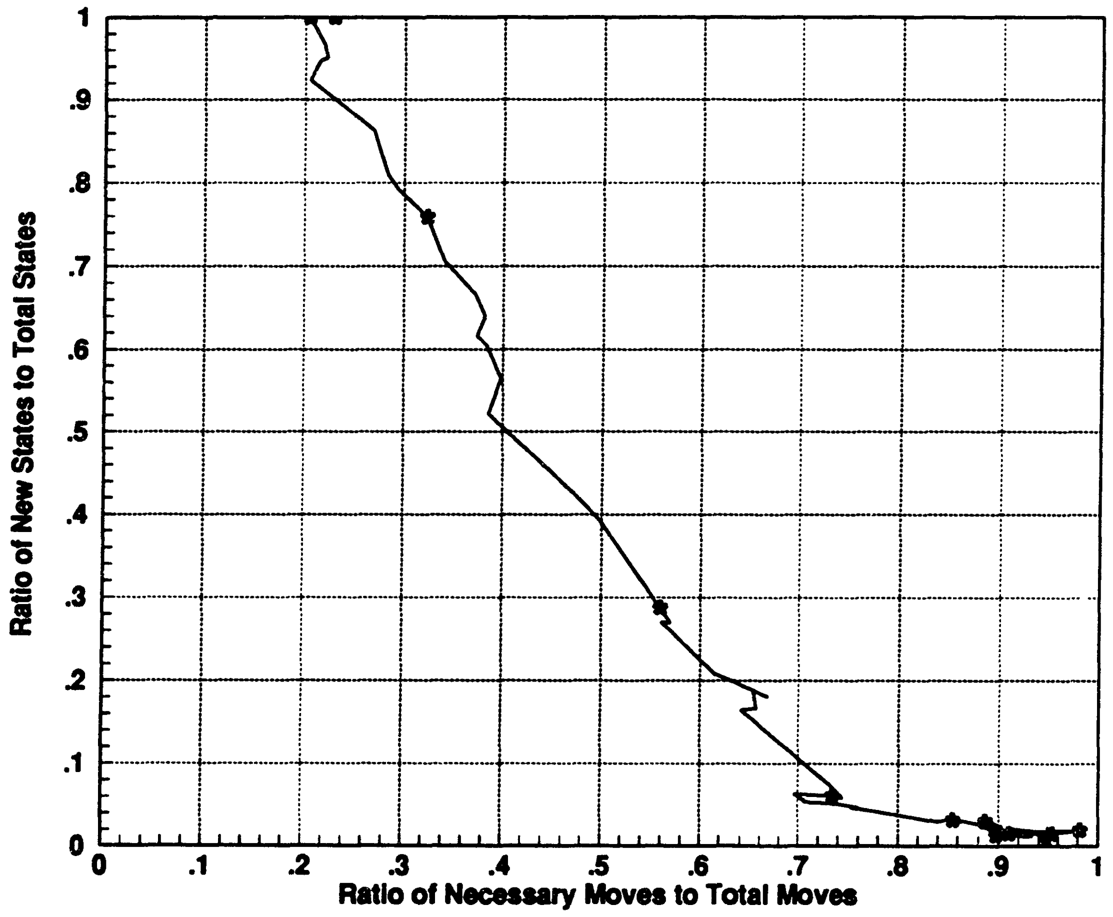

Figure 4.17: Learning Using Passive Wrist Compliance

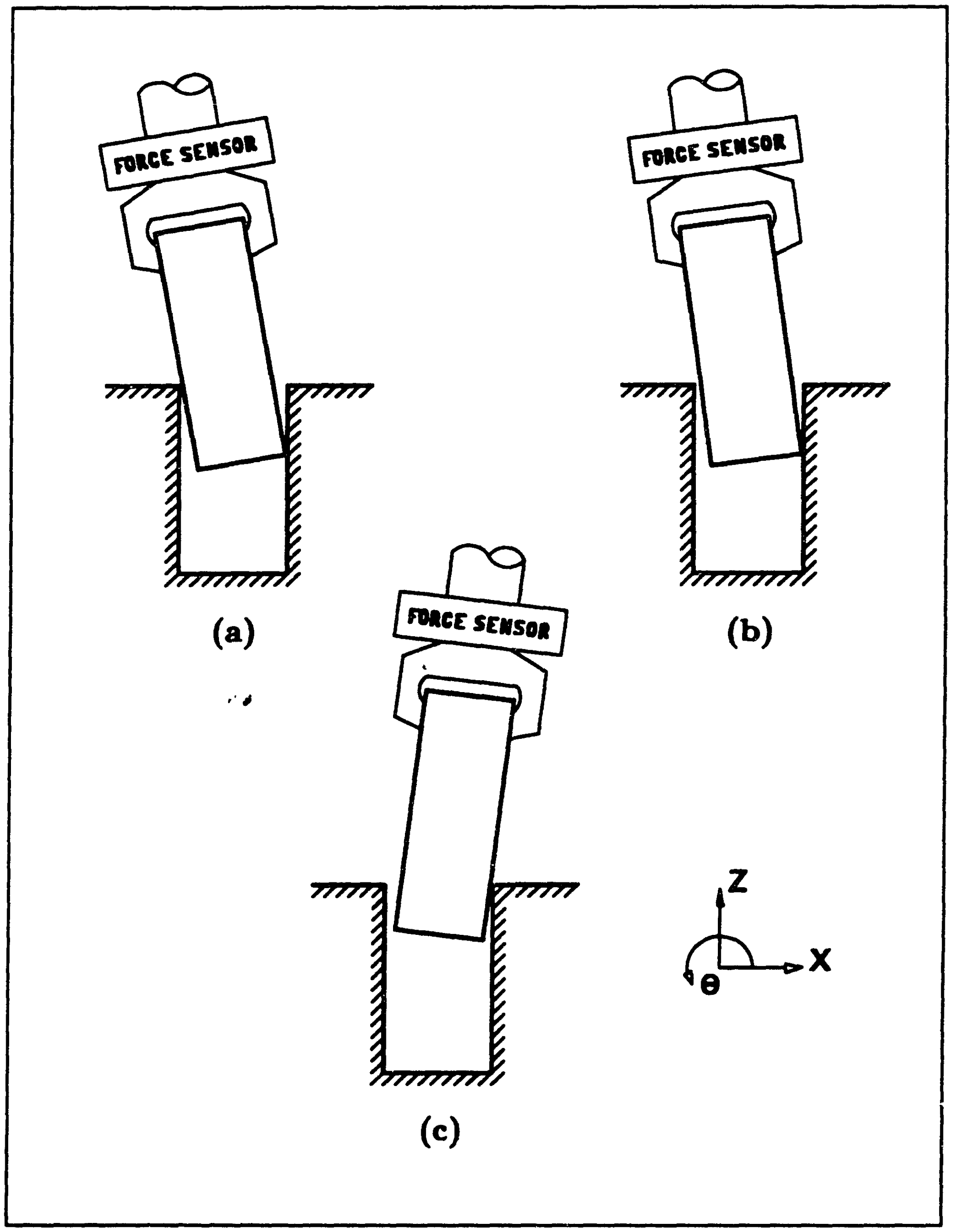

Figure 4.18: Pathological Contact Geometry for a System With a State Discretization of 2. A corrective move to the right in (a) results in one point contact (b). The configuration in (b) can not always be differentiated from configuration (c).

in the same state as shown in Figure 4.18c. Using the corrective move information associated with Figure 4.18c moves the peg in the wrong direction, but the system then recovers because two point contact has been reestablished and the next state arrived at (after incrementing in Z) will contain useful move information. The effect of this series of moves is not apparent until the next assembly is attempted that has a similar initial error. Because the assembly is completed by moving the peg to the right and rotating the peg clockwise, the moves stored in the data array for the condition shown in Figure 4.18c end up having roughly the correct magnitude, but the sign is wrong! If this series of events takes place in the lower of the two regions in Z, the next few assemblies contain some incorrect moves, but the system recovers. If the events take place in the higher of the two Z regions, the peg increments in Z until the force limit is exceeded and then proceeds to make all the wrong moves. This causes the hole to rotate out from under the peg and increase the rotational error to as much as 10 degrees. The peg then essentially jams in the hole. The learning system can be protected from this situation by having a sufficiently large number of saved moves (past history), but "sufficiently large" is a difficult number to pin down. Even if this number is chosen correctly, the resulting state information has such a large distribution that it is good for little more than determining the correct direction in which to move.

The distribution of move distances in a given state gets tighter as the discretization level of the states gets higher until the randomness in the force measurements due to friction and system errors become a significant part of the state size. This point was reached typically around a state discretization level of 5 to 10, giving an orientation accuracy of 1 degree in a system with initial errors of ± 5 degrees. Distributions this tight are necessary for good performance of approaches like the bold move strategy. See Section 4.1 for more detail on state resolution issues.

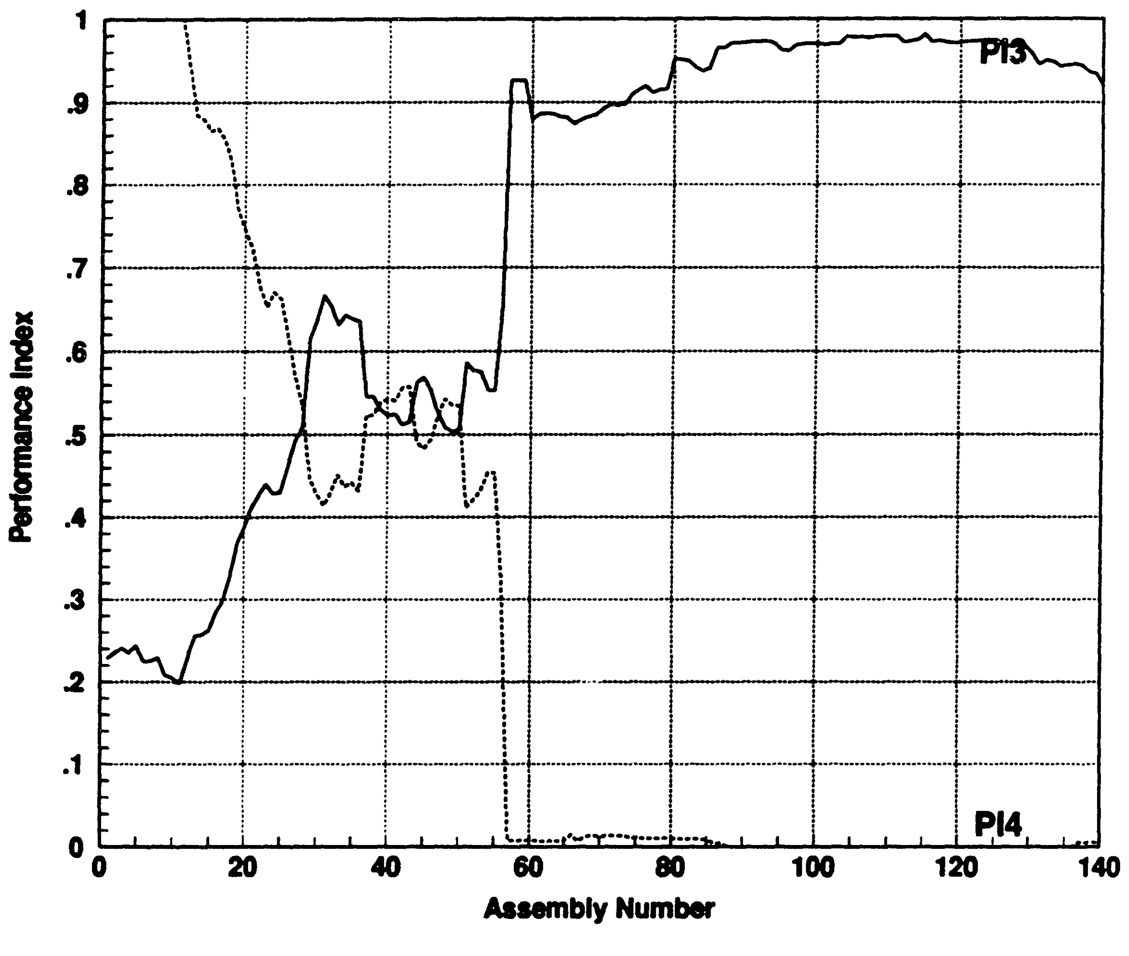

Figure 4.19: Convergence Using a State Discretization Level of 2

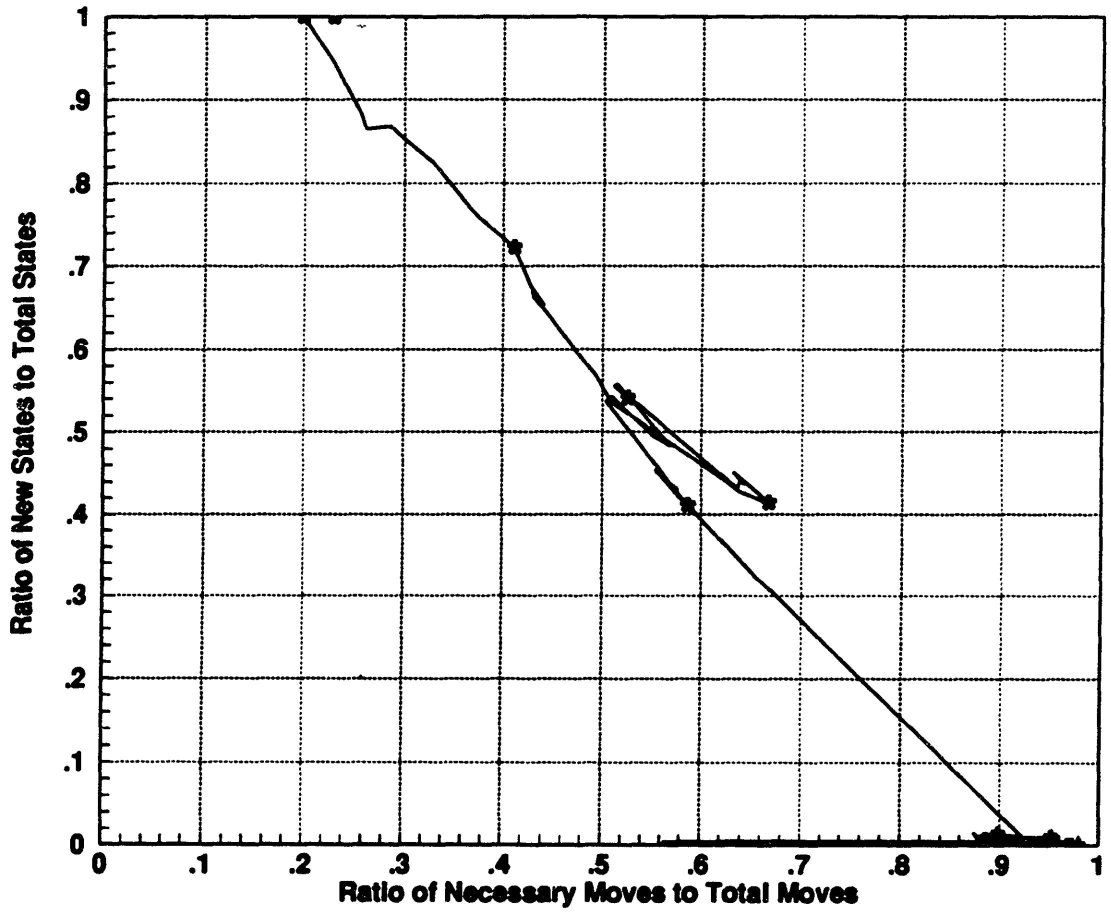

Figure 4.20: Learning Using a State Discretization Level of 2

4.4 Assembly of a One Inch Diameter Steel Peg using a clearance ratio of 0.005

The results of a series of assembly trials using a 3.0 inch long, 1.000 inch diameter steel peg and a 2.0 inch long, 1.005 inch in diameter aluminum hole are shown in Figures 4.21 & 4.22.

I feel that the poor behavior of the algorithm when the system was run with these parts was due almost entirely to the inability of the robot to respond consistently to requested moves of less than 0.005 inches or 0.002 radians. Many of the assembly attempts failed because the forces exceeded the limit of the force sensor. These runs were very frustrating to watch because the algorithm would typically correct for nearly all of the error before failing. The reason for this behavior is that the stiffness of the system goes up considerably as the peg is inserted into the hole. At the beginning of the insertion, the system stiffnesses were low enough so that the errors in the moves made by the robot did not result in the force sensor being overloaded. When the assembly was nearly complete, the system could be exceeding the force limit due to a large, positive F_θ value, and one *commanded* 0.002 radian corrective move in Θ could produce a large negative value of F_θ. The point in between that would have satisfied the FORCE-LIMIT constraint was passed over. A robot capable of more precise motion andor with more compliance in the system should be able to assemble these parts without difficulty.

4.5 Factors Limiting Assembly Speed

The algorithms developed in this thesis are not computationally intensive. *Essentially all of the time required to assemble a peg and hole was spent waiting for the robot system to settle.* This is not an uncommon result [Caine 1985]. An average

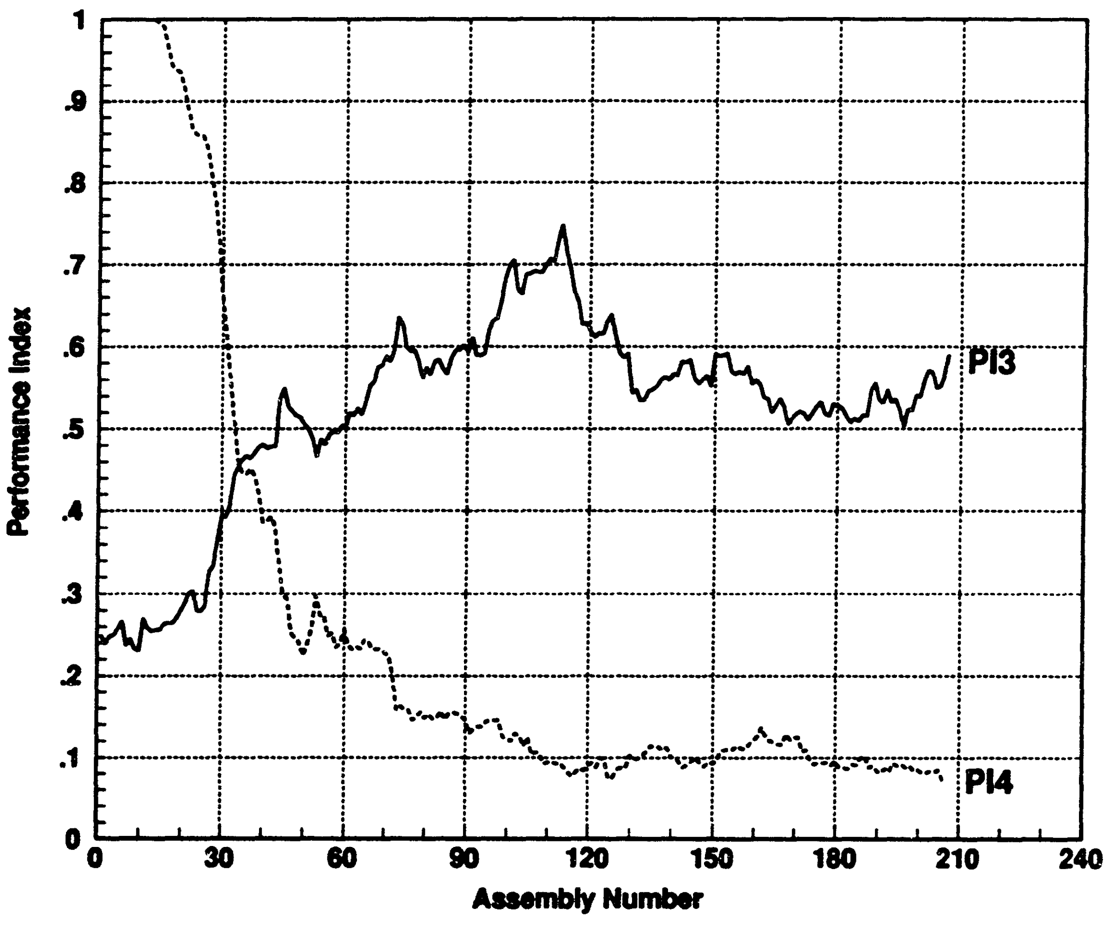

Figure 4.21: Convergence Using a Low Clearance Ratio (0.005)

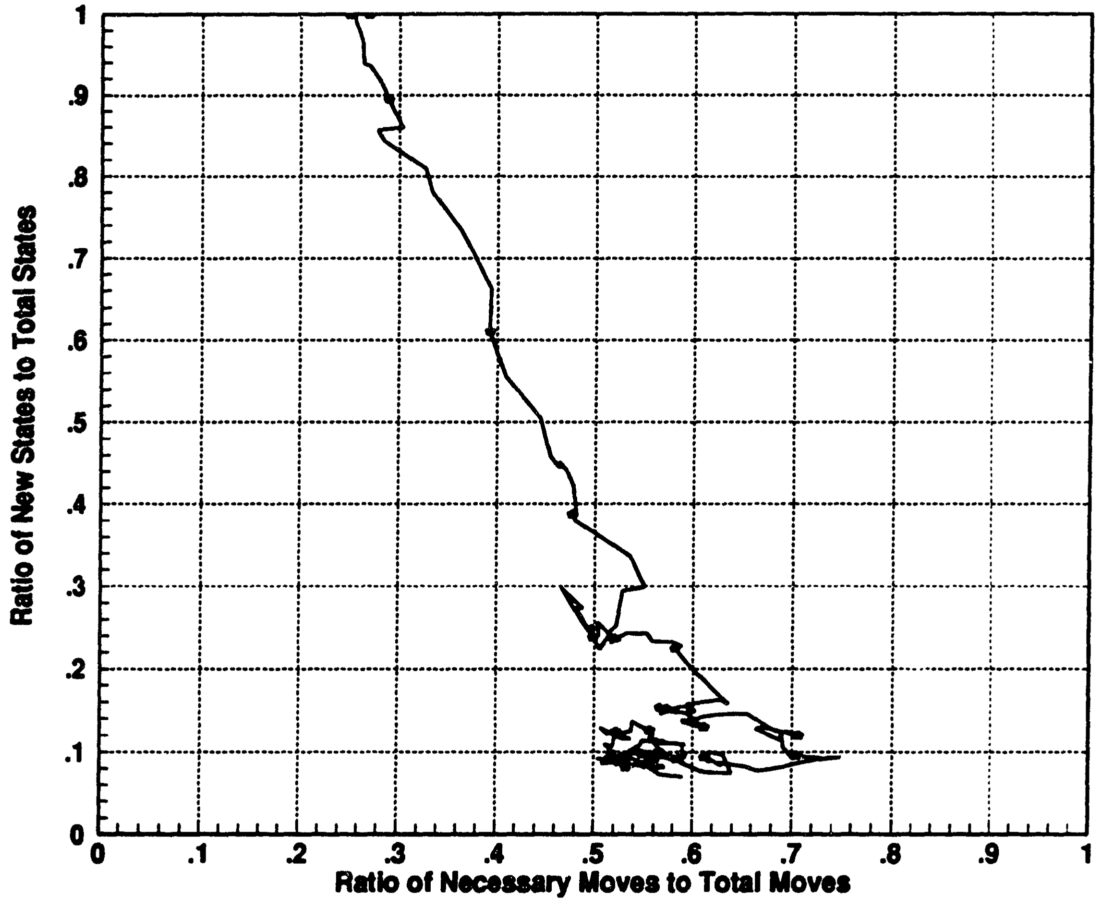

Figure 4.22: Learning Using a Low Clearance Ratio (0.005)

assembly required about 2 minutes during the early learning stages and about 20 seconds after the system had converged and was making nearly all of the correct moves. The computation time required to look up the move data averaged around 0.020 seconds per assembly. I have made some suggestions for improving assembly speed in Chapter 6.

Chapter 5

Conclusions

5.1 Summary of Research Contributions

This thesis makes the following contributions to the field of robotic assembly:

1. Force derivatives, calculated from changes in contact forces between the peg and hole due to incremental moves along the assembly path, are introduced as a source of information that allows the unique determination of the relative position of the peg and hole during one and two point contact.

2. Backtracking from the goal state through the moves made during an assembly is used to establish the relationship between the forces measured during the assembly and the position the parts were in when those forces were measured relative to the position of the parts in the goal state. This is effective because the relative position of the parts is known far more precisely at the completion of the assembly process than at the beginning.

5.2 Comparison with Compliant Motion

Compliant motion routines use geometric models of the assembly task in order to determine the combination of center of rotation and translational and rotational stiffness that minimize the insertion forces. Force sensor information, if used at all, is generally used to measure divergence from the anticipated force trajectories. The response to significant divergence is often to simply stop the assembly and wait for a human being to fix whatever is wrong.

Sufficient information is available from the force sensor, responses to recent moves, and results from previous assemblies so that the approach developed in this thesis does not require a geometric model of the parts or even of the assembly cell, i.e. the robot geometry, kinematics, stiffness, etc.. The only information about the assembly that is required beyond the sensor information is a nominal assembly path (NAP) and a termination condition. The NAP is the path that would be followed if the assembly was attempted open loop.

5.3 Necessary Conditions for Application of the Approach

For the approach I have developed in this thesis to be successful, the assembly cell, the parts, and the algorithm code must meet certain criteria:

- $\Delta F_x/\Delta Z$, $\Delta F_\theta/\Delta Z$, F_x, F_θ are a minimum set of system parameters that must be known (and be non-zero) in order for the relative position of the parts to be known (Test5, Test6). However, keeping track of other parameters can add robustness to the system, e.g. $\Delta F_z/\Delta Z$ is useful for sensing contact between the peg and the bottom of the hole.

- Repeatable, monotonic system and part stiffnesses are essential for my approach (I was surprised to discover that the Coke cans that I used for many of the tests I ran have non-monotonic stiffness when the contact force on the side of the can exceeds two pounds or so).
- There must be enough compliance in the system so that an incremental move along the NAP can be made that is large relative to the surface imperfections or the parts (without damaging the parts, damaging the force sensor, etc.)
- The geometry of the system must not change significantly due to the deflections resulting from the contact forces.
- The distance from the point(s) of contact to the point at which the forces are measured must be large.
- Large variations in system stiffness in different contact configurations should be avoided. A likely place for large changes to occur is at the transition from one to two point contact (Test7). Using different ranges and discretization levels for the states during one and two point contact help accommodate the transition, but the use of discrete moves in X, Z, and Θ make it impossible to determine exactly when the transition occurs, so the problem is not completely eliminated.
- The minimum discretization level is not well defined for parts that will not jam in that the peg will always eventually find it's way to the bottom of the hole, even at a discretization level of 2 (Test9). For parts that will jam, The discretization level must be high enough so that the angular error in the states never exceeds the angle at which jamming will occur.

I have developed a very simple machine learning algorithm that has successfully assembled several hard, smooth parts that can be modeled as a peg and hole. Convergence (visiting most of the states during a successful assembly and acquiring corrective move information) took approximately 20-50 assembly trials, or 500 state visits for a system with a clearance ratio of 0.04 and random orientation errors of up to ± 5 degrees. This implies that a real assembly system using these algorithms could converge quickly enough for the learning to be done on line. Learning by example or supervised learning would certainly improve the initial convergence rate. I did not pursue either of these approaches because I felt that they would make the learning rate even more difficult to define and measure. The present assembly time of 10-20 seconds is much too slow to be practical. For most applications, changes will need to be made in the algorithm that will allow it to accommodate the dynamics of the robot, or machines that have a much better behavior than the machine I used will have to be developed before this approach is worth considering for industrial use.

The robot that I used had some characteristics that limited the range of part geometries, sizes, and errors that the approach I developed could handle (see Section 5.5). The fundamental limit of the accuracy of the approach is a function of the surface finish of the parts and the distance that can be traveled with the peg and hole in contact without slipping or damage. For example, if a peg and hole have a 100 micro-inch finish and the robot can move 0.030 along the NAP with the peg and hole in contact before wedging or jamming occurs, the error in measurement of the contact angle will be up to arcsin(0.000200/0.030) = 0.38 degrees, even with perfect robot positioning and sensing. Interestingly, if the initial orientation error between the peg and the hole is small, e.g. 2-3 degrees, intentionally increasing the error before the assembly is started can actually improve the accuracy of the force derivative states (contact angle), because the peg can move much further along the NAP in one point contact. The optimum angle of contact is a function of all of the nonideal behavior of the system as well as the geometry of the parts and therefore should be determined experimentally, possibly as part of the learning algorithm.

5.4 Suitability of the Data Representation

Discrete representation of the system states and data appears to be a good choice for the peg in hole assembly task because friction introduces errors large enough so that the optimum resolution of the states in terms of information content can often be limited to a manageable number. Setting the range of the states by trial and error is slow and does not produce optimal results. The application of an automatic variable state resolution algorithm [Simons, et. al. 82] would improve the performance of the system significantly.

The one level search (Figure 2.6) used to find a combination of corrective moves is not adequate for assemblies where jamming creates some ambiguity in the choice of corrective move direction. This problem was pronounced during the assembly of small clearance ratio parts ($\approx .01$). Many assemblies were terminated because no combination of moves allowed by the algorithm reduced the forces enough for the assembly to continue.

5.5 Hardware Requirements

The restrictions placed on the assembly system behavior by the logic branching algorithms I developed are less stringent than are desirable for an approach that uses real time force feedback, but the restrictions are by no means loose. For various reasons, many combinations of system stiffnesses, minimum move size, force sensor resolution, gripper strength, peg and hole clearance, surface finish, stiffness, and strength (robustness) make the assembly algorithm I have developed unusable. Fortunately, there appear to be a reasonable number of systems and parts where my approach will at least be successful, if not practical. Desirable behavior for the assembly system is:

- Good backdriveability - The move size along the NAP that generates forces that are manageable by the gripper, part, and force sensor may be so small that the force measurements become sensitive to position errors and random noise in the environment. A stiff robot-part system is particularly sensitive. Controlling the system stiffness by adjusting the servo gains is far easier than changing the robot structure, especially if the stiffnesses need to be changed as a function of the state of the assembly.
- Low steady state position error - This relaxes the backdriveability constraint somewhat by allowing small moves to be made in a stiff or delicate system.
- No backlash - The force derivatives, which are the key to determining the relative orientation of the parts, require accurate *local* position information.
- High resolution position feedback - The tilt axis on the robot I used has a 4000 count/revolution encoder driven through a 15:1 transmission, giving 60,000 counts per revolution. Even with a relatively low gain set in the PD controller, 1 encoder count (0.0001 radians at the table) of position error appeared as a 0.4 inch-lb. torque. This shows up in the state resolution data presented in Section 4.1.

Notable by their absence from this list are restrictions placed on global position accuracy and repeatability.

Chapter 6

Future Work

3-D applications are perhaps the most obvious extension of the work presented in this thesis. Many real assembly tasks can be performed with the 2-D algorithms presented in this paper, however. The orientation errors of a peg that has been picked up from a pallet by a flat jaw gripper will generally be large only in the plane of the jaws. All other system errors are often small enough so that assembly is possible using a 2-D algorithm.

The peg and hole assembly is a particularly simple assembly task. The ability to use information from previous moves and assemblies potentially allows the logic branching approach to be extended beyond compliant motion to systems that generate locally ambiguous force information, such as systems where contact can be made perpendicular to the direction of motion. Heuristic search algorithms (wiggle the parts?) or probabilistic methods [Erdmann 89] could be used to get back on the path to the goal. Logic branching is a natural choice for these more complicated geometries. Systems using manipulators with many degrees of freedom and/or many sensors [Salisbury and Craig 82] have so far been much easier to build than to program. Logic branching may also be a good choice for these more complicated systems.

I am not at all sure that robots should mimic people in the way they assemble things, but pursuing this goal will at least lead to a better understanding of both systems (people and robots). The approach I presented in this thesis sets positions and then measures the forces and changes in forces that result from the robot moving to those new positions. A hybrid approach that can also set the forces and measure the changes in position due to the change in force is probably closer to what people do. Logic branching is a convenient way to jump from one mode to another in a hybrid algorithm.

The transition from one to two point contact generally has a significant effect on the size of the forces and force derivatives encountered during an assembly. Variations in system stiffness (often due to compliant parts or gripper) can also have an effect. The use of variable state resolution [Simons, et. al. 82] could be applied to this problem. Simons, et. al. proposed a system that would modify an initial 'guess' at the ideal state resolution by coalescing neighboring states that have the same move distances associated with them and dividing states that have an unacceptably large distribution of move distances. Retention of the actual force measurements along with the discretized values would allow the data to be used if the variable state resolution algorithm determined that a state's resolution was too coarse. Alternatively, a fuzzy set approach to the state resolution might be a good compromise between straight discretization and a functional representation of the states.

The approach presented in this thesis relies on the tolerances in the parts being small compared to the clearances for accurate determination of the relative positions of the parts during two point contact. If valid, this assumption means that changes in contact angle due to incremental moves along the assembly path are and accurate measure of both their relative position in Z and Θ. For many assemblies this is not a valid assumption. A possible solution to this problem is the use of higher order derivatives of the forces with respect to Z or the derivative of F_z/F_θ. Noise in these derivatives could make this approach impractical. Alternatively, an estimate of the relative position of the parts could be generated and carried through the assembly from beginning to end [Simunovic 79]. A goal of this thesis was to minimize the reliance on anything other than local force information for determination of the relative position of the parts. Using an estimate of the part orientation based on earlier estimates is not consistent with the goals of this thesis, but may be of great value for real assembly line applications.

More complex bold move strategies may be necessary for small and/or close fitting parts. Theoretically, the moves along the NAP and the corrective moves can be scaled to accommodate any size and shape of parts. Practically, non-ideal behavior of the robot places a limit on the minimum size of the parts. For example, once the parts have come into contact, only one or two moves along the NAP may be possible before wedging occurs. It may be beneficial to back up and repeat these moves several times in order to reduce the uncertainty of the force measurements.

Running the assembly algorithms under simple PD control is definitely not the best choice for use in industry. At the very least, a vibration reduction algorithm [Singer and Seering 87 88] should be used. Velocity control could eliminate most of the time needed for the system to settle when using incremental moves under PD control. Further extending this approach, safe velocity trajectories could be learned by the system, probably after the system learned the correct responses to most of the possible force states.

Learning the termination condition may be practical. Solution of this problem will likely be extendable to the broader application of learning how to set objects down. The present condition is intentionally conservative in that it requires a force in the Z direction to be significantly larger than any Z force expected during the assembly. The drawback to this high limit is the generation of new states or the corruption of existing states with force and torque information generated while the peg is contacting the bottom of the hole but has not yet exceeded the termination threshold. A reasonable way around this problem may be for the algorithm to learn the value of $\Delta F_z/\Delta Z$ that corresponds to the peg contacting the bottom of the hole. This parameter would then be used in conjunction with a simple force threshold on ΔF_z and a neighborhood of Z.

What to do if the average assembly path is different enough from the nominal assembly path so that there is a significant performance gain achieved by determining the difference? A straightforward solution is to use the NAP as a guide only and generate an rms (or other) fit to the data. This approach works well to correct discrepancies between the NAP and the real system due to static errors, e.g. a repeatable offset due to incorrect positioning of a part fixture. A more difficult problem to address occurs when the NAP changes orientation due to random initial errors in the orientation of the stationary part. Initial one point contacts between the peg and hole do not generate any force information that can be used to determine the true assembly path.

References

1. **Asada and Yousef-Toumi,** *"Direct-Drive Robots: Theory and Practice"*, MIT Press, Cambridge, MA, *1987.*

2. **Asada, H. and Kakumoto, Y.,** *"The Dynamic RCC Hand For Compliant Assembly"*, Procedings of the IEEE Conference on Automation and Control, Philadelphia, PA, *1988.*

3. **Boothroyd, G.,** *"Design for Assembly - A Designers Handbook"*, Department of M.E., U. Mass. Amherst, Amherst, MA, *1980.*

4. **Brooks, Rodney A.,** *"Solving the Find-Path Problem by Representing Free Space as Generalized Cones"*, M.I.T. Artificial Intelligence Laboratory Memo 674, Massachusetts Institute of Technology, Cambridge, MA, *May, 1982.*

5. **Brooks, Rodney A.,** *"Symbolic Error Analysis and Robot Planning"*, M.I.T. Artificial Intelligence Laboratory Memo 685, Massachusetts Institute of Technology, Cambridge, MA, *September, 1982.*

6. **Brooks, Rodney A.,** *"Planning Collision Free Motions for Pick and Place Operations"*, M.I.T. Artificial Intelligence Laboratory Memo 725, Massachusetts Institute of Technology, Cambridge, MA, *May, 1983.*

7. **Buckley, Stephen J.,** *"Planning and Teaching Compliant Motion Strategies"*, Ph.D. Thesis & M.I.T. A.I. Lab Technical Report 936, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, *January, 1987.*

8. **Caine, Michael E.,** *"Chamferless Assembly of Rectangular Parts in Two and Three Dimensions"*, M.S. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, *June, 1985.*

9. **Caine, Michael E.,** *"The Effect of Part Shape On Planar Insertion Operations"*, Procedings of the 2nd ASME Conference on Flexible Assembly Systems, Chicago, IL, *Sept, 1990.*

10. **Canny, John,** *"Collision Detection for Moving Polyhedra"*, M.I.T. Artificial Intelligence Laboratory Memo 806, Massachusetts Institute of Technology, Cambridge, MA, *October, 1984.*

11. **Connell, Margaret E. and Utgoff, Paul E.,** *"Learning to Control a Dynamic Physical System"*, Sixth National Conference on Artificial Intelligence, Seattle, WA, *pp. 456-460, July, 1987.*

12. **Donald, Bruce R.,** *"Error Detection and Recovery for Robot Motion Planning with Uncertainty"*, PhD. Thesis & M.I.T. A.I. Lab Technical Report 982, Department of Electrical Engineering and Computer Science , Massachusetts Institute of Technology, Cambridge, MA, *July, 1987.*

13. **Dufay, Bruno and Latombe, Jean-Claude,** *"An Approach to Automatic Robot Programming Based on Inductive Learning"*, *in* Robotics Research - The First International Symposium, The MIT Press, Cambridge, MA, *pp. 97-116, 1984.*

14. **Erdmann, Michael A.,** *"On Motion Planning with Uncertainty"*, S.M. Thesis & M.I.T. A.I. Lab Technical Report 810, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, *August, 1984.*

15. **Erdmann, Michael A.,** *"On Probabilistic Strategies for Robot Tasks"*, Ph.D. Thesis & M.I.T. A.I. Lab Technical Report 1155, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, *August, 1989.*

16. **Eppinger, Steven D.,** *"Modeling Robot Dynamic Performance for Endpoint Fore Control"*, Ph.D. Thesis & M.I.T. A.I. Lab Technical Report 1072, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, *September, 1988.*

17. **Franklin, Gene F. and Powell, J. David,** *"Digital Control of Dynamic Systems"*, Addison-Wesley, Inc., Reading, MA, *1981.*

18. **Gaston, Peter C. and Lozano-Perez, Tomas,** *"Tactile Recognition and Localization Using Object Models: The Case of Polyhedra on a Plane"*,

Memo 705, M.I.T. Artificial Intelligence Laboratory , Massachusetts Institute of Technology, Cambridge, MA, *March, 1983.*

19. **Gordon, Steven J.**, *"Automated Assembly Using Feature Localization"*, Ph.D. Thesis & M.I.T. A.I. Lab Technical Report 932, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, *December 20, 1986.*

20. **Grimson, W. Eric L. and Losano-Perez, Tomas**, *"Model-Based Recognition and Localization From Sparse Range or Tactile Data"*, M.I.T. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, *August, 1983.*

21. **Hogan, Neville**, *"Impedance Control: An Approach to Manipulation"*, Proceedings of The American Control Conference, San Diego, CA, *pp. 304-313, June 6-8, 1984.*

22. **Inoue, H.**, *"Force Feedback in Precise Assembly Tasks"*, AIM-308, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, *August, 1974.*

23. **Johnson, David G. and Hill, John, J.** , *"A Kalman Filter Approach to Sensor-based Robot Control"*, IEEE Journal of Robotics and Automation, *Vol. RA-1, No. 3, pp. 159-162, September, 1985.*

24. **Karnopp, Dean**, *"Computer Simulation of Stick-Slip Friction in Mechanical Dynamic Systems"*, Transactions of The ASME, *Vol. 107, pp. 100-103, March, 1985.*

25. **Kwakernaak, Huibert and Sivan, Raphael**, *"Linear Optimal Control Systems"*, Wiley-Interscience (John Wiley and Sons, Inc.), *1972.*

26. **Losano-Perez, T., Mason, M., and Taylor, R.**, *"Automatic Synthesis of Fine-Motion Strategies for Robots"*, International Journal of Robotics Research, *Vol. 3, No. 1, Spring, 1984.*

27. **Mason, M. T.**, *"Compliance and Force Control for Computer controlled Manipulators"*, IEEE Transactions on Systems, Man, and Cybernetics, *Vol. SMC-11, no. 6, pp. 418-432, June, 1981.*

28. **Michalsky, R. S., Carbonell, J. G., Mitchell, T. M.**, *"Machine Learning "*, *Vol. 1*, Tioga Press, *1983.*

29. **Michalsky, R. S., Carbonell, J. G., Mitchell, T. M.,** *"Machine Learning ", Vol. 2,* Morgan Kaufmann, *1986.*

30. **Michalsky, R. S., Carbonell, J. G., Mitchell, T. M.,** *"Machine Learning ", Vol. 3,* Morgan Kaufmann, *1990.*

31. **Michie, D. and Chambers, R.,** *"BOXES: An Experiment in Adaptive Control",* Ed. E. Dale and D. Michie Eds., *1968.*

32. **Miller, F. W.,** *"Design for Assembly: Ford's Better Idea to Improve Products",* Manufacturing Systems, *March, 1988.*

33. **Narasimhan, S., Siegel, D. M., and Hollerbach, J. M.,** *"Condor: A Revised Architecture for Controlling the Utah-MIT Hand",* IEEE International Conference on Robotics and Automation, *April 25-29, 1988.*

34. **Ogata, Katsuhiko,** *"Modern Control Engineering",* Prentice-Hall, Inc., Englewood Cliffs, NJ, *1970.*

35. **Pasch, Ken,** *"Methods for Choosing Actuators and Transmission Components for High Performance Machines and the Design and Testing of an Air Motor/Magnetic Particle BrakeHybrid Actuator",* Masters Thesis, Dept of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, *April, 1984.*

36. **Pasch, Kenneth A. and Seering, Warren P.,** *"On the Drive Systems for High Performance Machines",* Journal of Mechanisms, Transmissions, and Automation in Design, *Vol. 106-1, March, 1984.*

37. **Peshkin, Michael A.,** *"Programmed Compliance for Error-correction in Robotic Assembly",* Proceedings of the 3rd IEEE International Symposium on Intelligent Control, Arlington, VA, *August 24-26, 1988.*

38. **Raibert, M., and Craig, J.,** *"Hybrid Position/Force Control of Manipulators",* Journal of Dynamic Systems, Measurement, and Control, *Vol. 102, pp. 126-133, June, 1981.*

39. **Salisbury, J. Kenneth,** *"Active Stiffness Control of a Manipulator in Cartesian Coordinates",* 19th IEEE Conference on Decision and Control, Albuquerque, NM, *December 1980.*

40. **Salisbury, J. Kenneth and Craig, John J.,** *"Articulated Hands: Force Control and Kinematic Issues",* International Journal of Robotics Research (MIT), *Vol. 1 no. 1, spring 1982.*

41. **Schneiter, John,** *"Automated Tactile Sensing for Object Recognition and Localization"*, PhD. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, *1986.*

42. **Seltzer, Donald S.,** *"Tactile Sensory Feedback for Difficult Robot Tasks"*, Proceedings of The Robot VI Conference, *March 1-4, 1982.*

43. **Simons, J., Van Brussel, H., De Shutter, J., and Verhaert, J.,** *"A Self-Learning Automaton with Variable Resolution for High Precision Assembly by Industrial Robots"*, IEEE Transactions on Automatic Control, *Vol. AC-27, No. 5, October, 1982.*

44. **Simunovic, S.,** *"An information Approach to Parts Mating"*, PhD. Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA, *1979.*

45. **Singer, Neil C. and Seering, Warren P. ,** *"Controlling Vibration in Remote Manipulators."*, ASME Design Automation Conference., *Vol. 2*, Boston, MA., *pp. 11, September 27–30, 1987..*

46. **Singer, Neil C. and Seering, Warren P. ,** *"Using Acausal Shaping Techniques to Reduce Robot Vibration."*, Proceedings of the 1988 IEEE International Conference on Robotics and Automation., Philadelphia, PA., *April 25–29, 1988..*

47. **Smith, R.G., Mitchell, T.M., Chestek, R., and Buchanan, B.G.,** *"A Model for Learning Systems"*, Proceedings of the Fifth International Joint Conference on Artificial intelligence, Cambridge, Ma., *1977.*

48. **Strip, David R.,** *"Insertions Using Geometric Analysis and Hybrid Force-Position Control on a PUMA 560 with VAL II"*, Sixth National Conference on Artificial Intelligence, Seattle, Wa., *pp. 695-698, July, 1987.*

49. **Strip, David R.,** *"A Passive Mechanism for Insertion of Convex Pegs"*, Intelligent Machine Principles Division 1141, Sandia National Laboratories, Albuquerque, NM, *October 13, 1988.*

50. **Townsend, William,** *"The Effect of Transmission Design on Force Controlled Manipulator Performance"*, PhD. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, *April, 1988.*

51. **Vaaler, Erik, and Seering, Warren,** *"Automated Assembly with Systems Having Significant Manipulator and Part Location Errors"*, Proceedings of the 1987 IEEE International Conference on Robotics and Automation., Raleigh, North Carolina., *March 31-April 3, 1987.*

52. **Van Brussel, H. and Simons, J.,** *"Automatic Assembly By Active Force Feedback Accomodation"*, Procedings of the 8th International Symposium on Industrial Robots, *pp. 181-193, 1978.*

53. **Whitney, Daniel E. and Nevins, James L.,** *"Computer-controlled Assembly"*, Scientific American, *Vol. 238, No. 2, pp. 62-74, 1978.*

54. **Whitney, Daniel E.,** *"Quasi-static Analysis of Compliantly Supported Rigid Parts"*, ASME Journal of Dynamic Systems, Measurement, and Control , *Vol. 104-1, 1982.*

55. **Whitney, Daniel E.,** *"Historical Perspective and State of the Art in Robot Force Control"*, Proceedings of the IEEE Robotics Conference, St. Louis, MN, *March, 1985.*

56. **Whitney, Daniel E., Gustavson, Richard E., Hennessey, Michael P.,** *"Designing Chamfers"*, International Journal of Robotics Research (MIT), *Vol. 2, number 4, Winter 1983.*

57. **Whitney, Daniel E. and Rourke,** *"Mechanical Behavior and Design Equasions for Elastomer Shear Pad Remote Center Compliances"*, ASME Journal of Dynamic Systems, Measurement, and Control , *Vol. 108-1, 1986.*

Appendix A

Design and Construction of the Robot

This appendix is an extension of [Vaaler and Seering 86]. [Vaaler and Seering 86] was written before the servo systems were installed on the axes so the sections on the servo hardware and performance are new material. The robot was designed as a test bed for performing assembly tasks and as a vehicle for studying robot control strategies. Upon initiation of the project, we set out to design a manipulator which would be an order of magnitude stiffer (first structural mode greater than 50 Hz with a combined axis weight of 100 lbs) and an order of magnitude quicker (peak acceleration greater than 3g) than commercially available robots of comparable size.

Our robot was designed with a rather unorthodox geometry (Figure 1.2). Two axes of the wrist are attached to the base rather than to the arm. The configuration of these two axes is similar to that of a standard two axis welding table. Most six degree-of-freedom robots are designed with their axes in series. Because the weight of the wrist axes must be carried by the other axes, the wrist axes are usually much less stiff than the axes further back in the kinematic chain. By mounting the wrist to the base, we were able to make the two wrist axes very stiff without compromising the performance of the other axes.

Choice of this configuration enhances the robot's ability to perform some types of work on a component. The component is mounted to the wrist base from where it can be oriented with two degrees of freedom. The robot's arm moves with three linear cartesian and one rotary degree of freedom. The two combined provide the six degrees of freedom necessary to allow the robot to access any position of the workpiece. Conventional anthropomorphic robots generally loose effective degrees of freedom as they reach to the back sides of workpieces. They actually have complete use of six degrees of freedom throughout a relatively small portion of their specified workspace. The four-plus-two configuration does not have this problem. It does, however, have it's own set of disadvantages. The workpiece must move in order to access it, so the position of the workpiece is in general known less accurately with this axis configuration. Also, the ability to set washers, springs, etc. in place and rely on gravity to keep them there until later in the assembly process may be lost.

A.1 Hardware Design

Design of this robot began with the following set of primary system specifications:

1. *The workspace should be approximately 12 inches by 24 inches by 18 inches.* We felt that this workspace was large enough to accommodate may parts that robots might be asked to assemble. Since enlarging the workspace has a

negative influence on almost everything; eg. stiffness, accuracy, and weight; system optimization strongly favors minimizing the workspace size.

2. *The weight of the translating parts should not exceed 100 lbs.*

   100 lbs. was not a strict constraint. Rather it was used as a starting point for the axis design optimization. If the system stiffness resulting form this constraint was not acceptable, this constraint would have been relaxed. This limit includes the mass of two DC servo motors capable of producing the specified acceleration and speed.

3. *The position repeatability should be ±0.002 inch.*

   This goal was chosen rather arbitrarily, and, as we will discuss later, we were unable to meet it.

4. *The axes should be capable of 200 inches/second slew rates.*

   We did not want to be slew rate limited by our choice of bearings. 200 inches/second was comfortably above the maximum slew rate of any actuators that we considered using. Actuators capable of producing 4 G accelerations can cause the robot to reach 200 inches/second while traversing it's workspace. We have designed and evaluated a pneumatic servo system capable of accelerations greater than 5 G [Pasch 84].

5. *The stiffness of the axes should be maximized.*

   Not enough is known about how robot stiffness influences performance of various assembly tasks to set a particular stiffness spec. This is really an open-ended goal rather than a constraint.

6. *One horizontal dimension of the frame should be less than 32 inches.*
   Keeping one dimension of the robot frame under 32 inches allows the machine to pass through a standard door without being disassembled. This constraint limited range of motion in one direction.

7. *The upper frame should be supported by two columns.*
   The goal here was to maximize workspace accessibility. Because the robot is to be used in a research environment, we felt that we would need mechanical access for parts feeders, tool feeders, and instrumentation as well as visual access for video cameras and demonstrations.

8. *The cost of the mechanical hardware must be less than 12,000 dollars.*
   This was our budget. The design should be practical. We were interested in creating a design, components of which might find their ways into commercially produced systems. As a result we avoided the use of esoteric technologies.

The following constraints were derived from those described above.

1. *The bearings will be made with cam followers traveling along hardened ways.*
   Most standard linear bearings can not meet our 200 in/sec slew rate spec. Table 1 lists the maximum recommended slew rates for the standard bearings we considered. The maximum speed rating of the ball bushings was high enough so that we did consider them seriously. We determined that 2 in. diameter, continuously supported ways would be required to produce a system that had the load capacity and stiffness of the much smaller cam followers and ways that we did use. The large envelope of the ball bushing assemblies, the indeterminant nature of their preload, and the difficulty in

machining the supports for 4 bearings on each axis were enough to convince us to use cam followers. Had we not placed such importance on the stiffness constraint, ball bushings might have been a better choice than cam followers. It might appear from this list that cam followers are the only bearings that meet our slew rate spec. This in not quite true. There are several *non-standard* bearings that do meet our spec. Special order "Tychoway" type bearings are available that will meet our slew rate spec, but the price we were quotcd was more than an order of magnitude higher than for cam followers. Tycho-way bearings, in addition to their high cost, are much more difficult to set up than cam followers. Gas and oil bearings will easily meet our specs for speed and repeatability. They do have some significant drawbacks, however. Gas bearings, if they are operated on shop air (100 psi), must be quite large or have small clearances to be as stiff as the cam followers we used. Reasonable clearances (0.001 inch - 0.002 inch) require unreasonably large bearing surfaces (30 square inches). Reasonable sizes (1-4 square inches) require unreasonable tolerances (0.0001 inch - 0.0002 inch). The use of an air amplifier is an alternative. Supplying the bearings with 2000 psi air would allow the use of practical bearing sizes and clearances. Construction, setup, and maintenance costs would be high. Gas bearings might be justified if it was determined that coulomb friction could not be tolerated in the axis motion. Oil bearings have their own set of drawbacks, the most significant being cost and cleanliness.

2. *The cam followers must be 0.625 inch wide.*

   The next larger size of cam followers (0.750 inches wide) are stiffer than the 0.625 inch followers we used, but the larger, heavier ways they required would have consumed half of the weight allotted for the moving parts. The load capacity of the next smaller size (0.500 inch) was not large enough to meet the combined acceleration and moving weight constraints. We followed recommended practice by preloading the cam followers slightly in excess of the expected dynamic load (to prevent skidding), so some of the bearing loads are always at least twice the load imposed through the structure.

3. *The bare axes could weigh up to about 25 lb.*

   We determined how long and therefore how heavy the bearing ways had to be in order to span the workspace. We also made an approximation for the weight of the X and Z axis actuator systems that would be required to meet the acceleration and slew rate specs (The Y axis actuator is stationary). We were left with only 25 lb. for the bare axes.

The constraints just presented set up the problem of optimizing dimensions of the moving axes. Because our goal was to build a system that was stiff and fast, we evaluated changes in geometry on the basis of the following ratio:

$$\frac{\text{increase in mass}}{\text{decrease in endpoint deflection}} \tag{A.1}$$

For an ideal design, this ratio would be the same for all structural components. We considered three sources of deflection in our analysis: beam bending, beam shear, and cam follower deflection. The deflections due to torsion and tension are negligible. The optimization was done based on the assumption that the robot would spend most of the assembly time working around the middle of the worksurface with the Z axis almost fully extended. We made all of the deflection calculations based on a 100 lb load being applied to the end of the Z axis in the X and Y directions. The "total" deflection is the deflection measured at the end of the Z axis in the direction of the force. Using a fixed value of 100 lb (approximately the maximum expected load) gave us a better feel for the way the system was behaving than if we had used a symbolic constant. Rough calculations indicated that the cam followers would be the first elements of the structure to fail if the system was overloaded, so we generated the "optimum" axis shapes and then checked the maximum expected stresses in the axes. Figure A.1 is a plot of system characteristics versus the variation of the spacing of the cam followers that support the Z axis.

This type of plot shows the behavior of the equations used for selecting dimensions of the X, Y, and Z axes. As the cam followers are spaced farther apart, the X and Z axes must change configuration. This results in an increase in system weight. Based on the information in this and corresponding figures, a spacing of approximately 8 inches was chosen for the cam followers supporting the Z axis.

A.2 Construction Details

We considered building the moving axes out of wrought aluminum, cast aluminum, steel and composites. We felt that wrought aluminum was the best choice for our robot. This is because the machine we built is a prototype. If several robots of this general design were built and there was no expectation of modification, cast aluminum or composites would probably be a better choice of materials. There were significant modifications made to the axes during construction. We expect that there will be more changes made in the future. A welded wrought aluminum structure is much easier to modify than either a casting or a composite structure. We chose aluminum over steel because using aluminum results in a stiffer structure

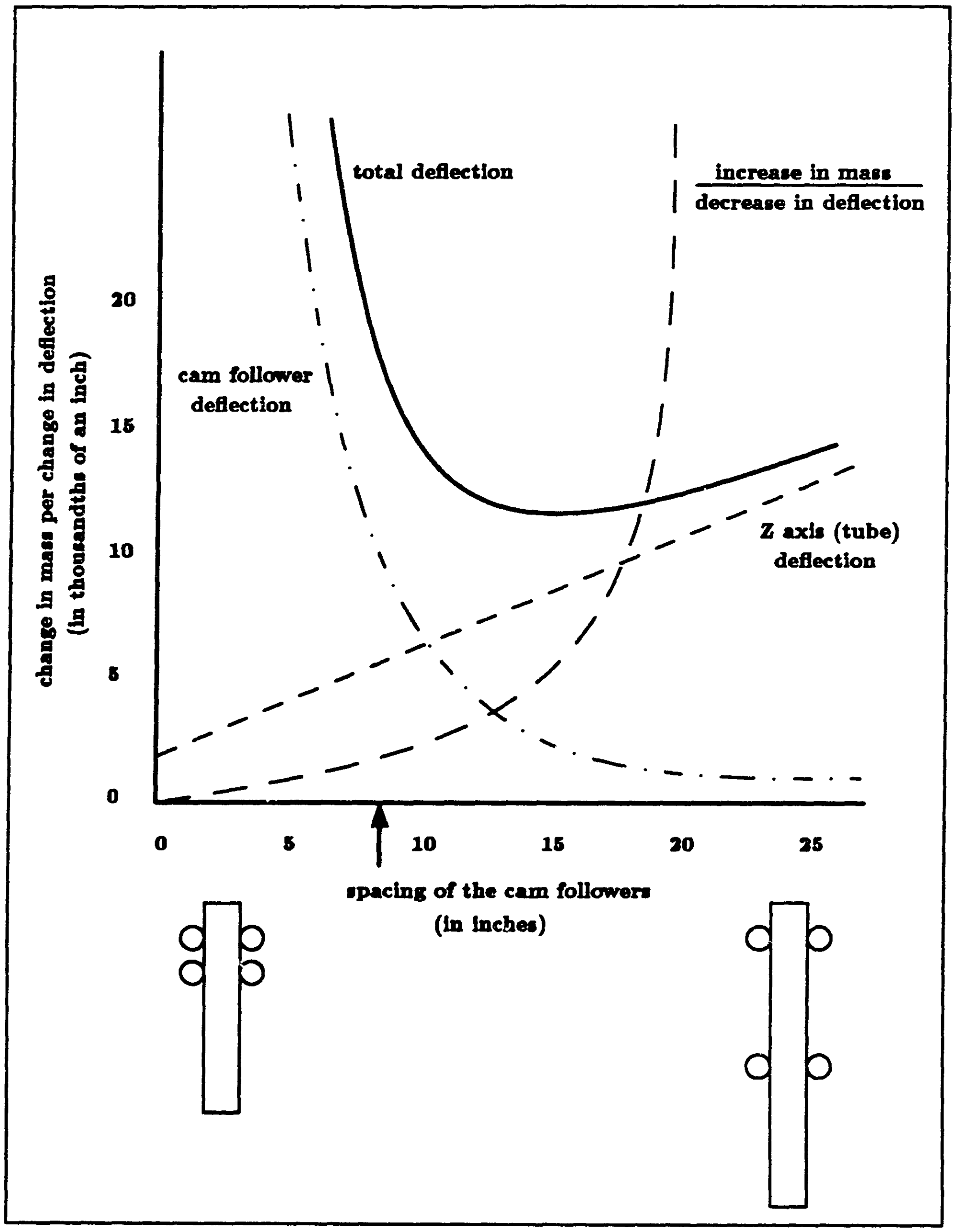

Figure A.1: Plot of the Variation of Component Deflections as a Function of Cam Follower Spacing on the Z Axis

(in this case) and is much easier to machine. Welding together a structure from steel that is one-third the thickness of the aluminum in an otherwise identical structure does not always produce a structure with similar stiffness. Although the axial stiffness of the structural elements may be roughly the same, the bending stiffness of the thicker aluminum elements would be much higher. Also, any warpage during welding can significantly reduce the stiffness of the structure. Warpage is much more likely to occur with the thinner steel structure.

The frame of the robot is built from standard structural steel C-channel. The base is welded from 12 in. C-channel. The top and bottom of the base were Blanchard ground parallel and flat to within ± 0.002 inch. The worksurface is Blanchard ground 3/4 inch mild steel plate. The columns are 6 inch by 6 inch box sections welded from 6 inch ship channel. The areas where bolts connect the frame sections are reinforced with 1 inch steel plate. These mounting surfaces were also ground flat. The upper frame is a 6 inch by 3 inch rectangular section tube welded from two pieces 6 inch C-channel. The mounting plates are again made from 1 inch plate. The top and bottom of these sections were Blanchard ground. The worst case frame stiffness is 50,000 lb./inch. The first mode of the frame is around 48 Hz with the frame resting on the floor.

The moving X, Y, and Z axes are welded, monocoque structures made from 0.125 inch wall aluminum. The axes roll on crowned cam followers so the parallelism of the cam follower mounting holes is not terribly critical. The cam followers are attached to the axes with eccentric studs. This allows the cam follower preload to be adjusted and compensates for considerable machining errors. The cam followers are a standard product of McGill Precision Bearings. Each axis is supported by 16 cam followers (Figure A.2). The followers ride on hardened ways that are ground flat to ± 0.0002 inch. The X and Y axis power systems are presently DC servomotor driven precision ball screws. The motors are attached to the screws with split shaft couplers to maximize actuator stiffness. The ball screws are accurate to within ± 0.0002 inch. The Z axis is driven by DC servomotor acting through a rack and pinion. The Z axis is hollow (4.00 inch diameter by 0.125 inch wall aluminum tubing), so air, power, and sensor cables can easily be routed through it (Figure A.2). The travel in the cartesian axes workspace is approximately 12 inches x 22 inches x 16 inches (XYZ).

The Spin axis is mounted on the end of the Z axis. It was built around a standard 1 inch 5C collet and collet block. The Spin axis is directly driven by a samarium-cobalt torque motor. The motor has a continuous torque rating of 2.5 ft-lb. The peak rating is around 5 ft-lb. The collet block is mounted in a pair of extra light series angular contact bearings. The collet is actuated by a pneumatic piston that applies 400 lb draw. The 1 inch collet serves as a "universal" end effector interface that can change end effectors automatically. The axis also contains a pair of slip rings and a set of rotating seals to supply power and air to the end effector. This arrangement results in a wrist that is capable of continuous rotation. All cabling is routed through the Z axis.

Samarium-cobalt torque motors rated at 3.0 ft-lb. (continuous) drive the axes of the wrist or positioning table through spur gears. The Tilt axis is mounted in preloaded deep groove ball bearings. The Pan axis is supported by a 4 inch i.d. Kaydon 4-point contact thin section bearing. The large bore of the bearing allows convenient access from beneath to the worksurface of the table for electricity, air, and sensors.

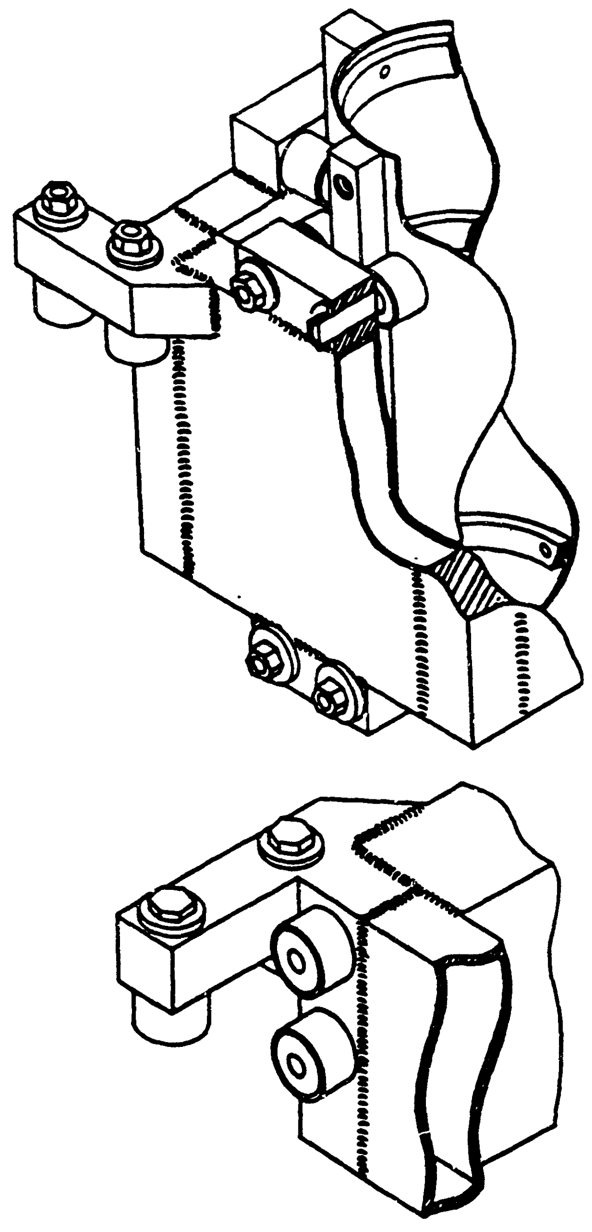

Figure A.2: Construction Details of the X, Y, and Z Axes - The top drawing shows the Z axis tube cut away to display the rings used for internal stiffening and way attachment. The Y axis in the top drawing and the X axis in the bottom drawing are cut away to show their monocoque construction.

A.3 The Servo System

We chose the motors primarily for their power to weight ratio and actuator compatibilities. The X and Y axis motors have a maximum rpm rating of 3500. This matches up well with the 3000 rpm peak rating of the ball screws. Transmissions, ball screws for the X and Y axes and a rack and pinion for the Z axis, were chosen according to the guidelines presented by [Pasch and Seering 84b].

The goal of the servo loop design was to do the minimum necessary to make the axes well behaved for use with the learning algorithms. The learning algorithms do not require accurate absolute position information, nor do they require accurate position control, so the demands placed on the controllers were not unreasonable. Simple PD controllers were found to adequate for all of the axes. An error between commanded position and encoder position of 0.002 inch or 0.002 radians was somewhat arbitrarily chosen to be acceptable. The gains required to achieve 0.0005 inch servo error (2 encoder counts) tended to generate some limit cycling. In order to meet this spec, the proportional gains had to be set quite high in order to overcome the erratic levels of coulomb friction introduced by the preloaded cam followers. In order to make sure that the controllers were robust, the gains were then turned up until the influence of higher modes became noticeable in the form of limit cycles and longer settling times. We were able to increase the gains by at least 30% on all of the axes which we felt was sufficient to guarantee acceptable behavior in all expected operating conditions. The controllers were developed using MatrixX, a software package from Integrated Systems Inc. (ISI). State estimators were used because optical encoders were the only source of feedback. LQG design techniques were used to determine the estimator poles. [Ogata, Kwakernaak and Sivan, Franklin and Powell].

The servo systems for the robot use the following hardware:

1. The X axis is driven by an Aerotech #1410 servo motor through an NSK ball screw with a 1 inch lead. The servo amplifier is a Copley Controls #240.

2. The Y axis is driven by an Aerotech #1410 servo motor through an NSK ball screw with a 1 inch lead. The servo amplifier is a Copley Controls #240.

3. The Z axis is driven by an Aerotech #1410 servo motor through a 20 pitch rack and pinion. The pinion has a pitch diameter of 1 inch. The servo amplifier is a Copley Controls #240.

4. The Spin axis is directly driven by an Inland frameless torque motor #QT-3102-A. The servo amplifier is a model #4020 from Aerotech.

5. The Pan axis is driven by an Inland frameless torque motor #QT-2404-A through a 20 pitch spur gear reduction of 10 : 1. The servo amplifier is a model #4020 from Aerotech.

6. The Tilt axis is driven by an Inland frameless torque motor #QT-2404-A through a 20 pitch spur gear reduction of 15 : 1. The servo amplifier is a model #4020 from Aerotech.

A.4 System Performance

The X and Y axes of the robot are each capable of about 2 G acceleration. Because it does not have to carry the weight of the X and Y axes, the Z axis can accelerate at more than 3 G unloaded. The first natural frequency of the robot when it is not attached to the floor is around 50 Hz. When bolted to the floor, the robot has a first mode of 12 Hz. This mode is comprised essentially of the robot moving as a rigid body with the floor (75 lb/ft load rating) acting as a spring. The worst case structural stiffness of the cartesian axes is 4000 lb/inch. The rotary axes have a minimum off axis stiffness of 5000 lb/inch and 3000 ft-lb/degree. On axis (servo) stiffness is considerably lower. The combined structural stiffness of the 6 axes is approximately 2000 lb/inch and 1000 ft-lb/degree. This system stiffness is at least an order of magnitude higher than that of most commercially available robots of comparable size. Because of dimensional problems with the cam followers, we were unable to meet our ± 0.002 inch repeatability spec. The dimensional tolerances of the cam followers are loose - up to 0.001 inch TIR. Our solution to this problem was to buy more cam followers than we needed and select the best of the lot. All of the followers we used had a runout of less than 0.00035 inch. Absolute uncalibrated system accuracy was ± 0.010 inch throughout the robot workspace [Podiloff 85]. Repeatability was ± 0.006 inch. Had we calibrated the workspace, we could have improved accuracy by 0.004 inch. High accuracy is important for us because we intend to produce programs for this robot off line without using a teachbox. Evaluation of the system's performance to date supports the concept of including high stiffness in the design specifications. High stiffness facilitates rapid, precise movement. High stiffness also appears to play an important role in improving dynamic performance when the system is operating under alternative control modes such as closed loop force of stiffness control.

We are reasonably satisfied with the robot we built. It does nearly everything we called for in the original specs. Time will tell if it is well suited for assembly work. However, when we build another one, we would certainly make some changes. Most importantly, we would make the workspace smaller. Most assembly tasks can be performed within a 1 foot cube. Most robots have larger workspaces simply to allow for accessing parts from many feeders. We believe that feeders should be designed to deliver parts to the assembly robot rather having the robot reach for them. If this were done, the robot could be made to be significantly smaller, and hence faster, stiffer, and cheaper.

A number of design details deserve reevaluation. The connection between the columns and the upper frame is a poor design. There is presently no good way of adjusting the angle between these two components. Also, we were not able to locate a shop that could easily grind the two mounting surfaces on the columns square with each other. Redesigning this mounting surface to lie in the XZ plane would probably solve both of these problems.

We would change our fabrication techniques slightly. All of the frame components were stick welded. There is a considerable amount of distortion in the columns and the base. Most of the welds should have been done with a TIG machine. Careful fitup and shallow penetration would have minimized distortion while having adequate strength. The extra welding cost would be recovered in the reduced grinding costs. The moving axes should have been rehardened after welding. 6061 aluminum is gummy after being welded (annealed). This made some of the machining quite slow and difficult. The next set of motors we use on the robot will have their endbells machined into the motor mounts. This saves weight, increases stiffness, and reduces tolerance buildup. The motor mounts are separable form the axes and are intended to be used only with one specific motor, so little flexibility would be lost.

Appendix B

The Computer Hardware

Three separate processors run simultaneously and divide the work of controlling the robot. They share a common backplane which has a complement of interface cards to connect to the outside world. A separate computer provides a development environment and data storage.

The computer system is comprised of a Sun 3/180 Unix Workstation connected to a VMEbus expansion box. The expansion box holds a system controller, three single board processors, a digital to analog converter board, an analog to digital converter board, five optical encoder reading cards, a digital I/O board, and extra memory. The backplanes of the Sun and the VMEbus expansion box are connected together so that data can be transferred to and from the Unix system.

The Sun 3/180 is used solely as a convenient development environment, it does not deal with any of the robot control because Unix is not a real-time system. All robot control is done by the single-board processors located on the VMEbus. The software for the Sun (described in the next section) provides a simple user interface to the processor boards, runs the software compiler, and provides space for permanent data storage. This section contains a description of the boards in the system I used to perform the experiments described in this thesis.

1. Ironics IV–3273 System Controller:
   All VMEbus systems require a board that mediates bus access and provides functions such as a system reset and clocking.

2. Ironics IV–3201A Processor:
   A single board, self-contained microcomputer. It is based on a 16 MHz Motorola 68020 processor with a 68881 floating point coprocessor and one megabyte of dynamic RAM. The Ironics processor card has several nice features, including a mailbox for interrupt driven communication between processor cards and dual-ported RAM that allows one processor card to access the memory of another processor card without interrupting it. The control system has three of these boards.

3. Motorola MVME 340A Parallel Interface/Timer Module:
   The MVME board provides digital I/O for the robot system. It has 50 I/O lines that can be used independently or in blocks of either 8, 16 or 32 for parallel data transfer. In addition, there are 8 lines for handshaking, 6 lines for timing functions and three 24 bit timers.

4. Data Translation DT1401 Interface Card:
   The DT1401 is 12 bit A/D converter. It has 32 channels of A/D which can either be used as 32 single-ended channels or as 16 differential channels. The input range is either 0 to 10 volts or ±10 volts. However, the board can prescale the input by a factor of 1, 2, 4, or 8 which gives it an effective range of either 1.25, 2.50, 5.00, or 10.0 volts. Each channel takes 10 microseconds

to sample and 15 microseconds to convert a reading which gives an overall sampling frequency of 40 kHz. Additionally, the DT1401 has two 12 bit D/A converters that can be configured to be either unipolar or bipolar, 5 or 10 volts maximum. And the DT1401 has 16 lines of digital I/O.

5. Data Translation DT1406 DAC Card:

   The DT1406 is a 12 bit D/A converter. It has 8 channels of D/A which can be set for either 0 to 10 volt operation, or −10 to +10 volt operation. The DT1406 is used to send control signals to the amplifiers.

6. Motorola Memory Board:

   This board contains two megabytes of dual ported RAM that is memory mapped onto the VMEbus. The memory available on the Ironics boards is already adequate for robot control; this memory exists to facilitate communications with the Sun. It provides a convenient place to store large quantities of data.

7. Whedco Dual Channel Incremental Encoder Interface Card:

   Each Whedco board provides two channels of encoder interface. The board accepts single ended or differential signals, can be configured to provide power to the optical encoders at 5 or 12 volts, and can be set for 1, 2, or 4 counts per line on the encoder. Position tracking is 32 bit, either 0 to 4,294,967,295 or ± 2,147,483,648 counts.

Appendix C

The Computer Simulation

I felt that there were so many interacting variables in the assembly problem that having control of them in a simulation was necessary. The goal of the simulation was not to develop an extremely accurate model of the assembly process but rather to build a tool that was accurate enough to generate qualitative information about the behavior of the various assembly algorithms under the influence of different system parameters such as friction, stiction, axis stiffness, and sensor errors. The computer simulation represents the geometry of an idealized, 2 dimensional peg and hole and includes the compliances of the robot.

C.1 Modeling Forces

The contact forces between the peg and the hole are calculated based on the amount of overlap (if any) of the boundaries of the peg and the hole and stiffness of the axes. For a simple peg and hole, there are only 4 points that need to be checked for contact: points a, b, c, and d in Figure C.1a. The contact forces are determined by calculating the distance from each of the possible contact points to the nearest boundary of the other part. If the distance is positive (no overlap) the force is zero. If the distance is negative, the distance and the system stiffnesses in X and Θ are used to calculate the contact force. The overlap and the associated forces are shown, with the overlap greatly exaggerated, in Figure C.1b.

A physical model of this approach is a completely rigid robot and hole with a peg made up from a rigid central core and a compliant cover. All deflection then occurs in the peg, at the point of contact. For small deflections, this method is far simpler and only slightly less accurate than attempting to more closely model reality by allowing no part overlap and distributing the deflection throughout the system.

C.2 Modeling Friction

Using a deterministic value for the effect of friction is adequate for simulating many systems [Karnopp 85]. Using an estimate of the statistical properties of the variation in system behavior due to friction is adequate for many more. LQG (Linear Quadratic Gaussian) control theory is a good example [Kwakernak, Sivan 72]. In LQG design, friction (and many other sources of nonideal behavior) are modeled as noise with a particular power spectral density. Unfortunately, no such body of knowledge exists for assembly systems. In an attempt to be as accurate as possible, I attempted to introduce the effects of friction into the system when and where nature does - at the parts, at every move. I used a Gaussian distribution with the tails clipped at 2 sigma. Figure C.2 shows the superposition of the

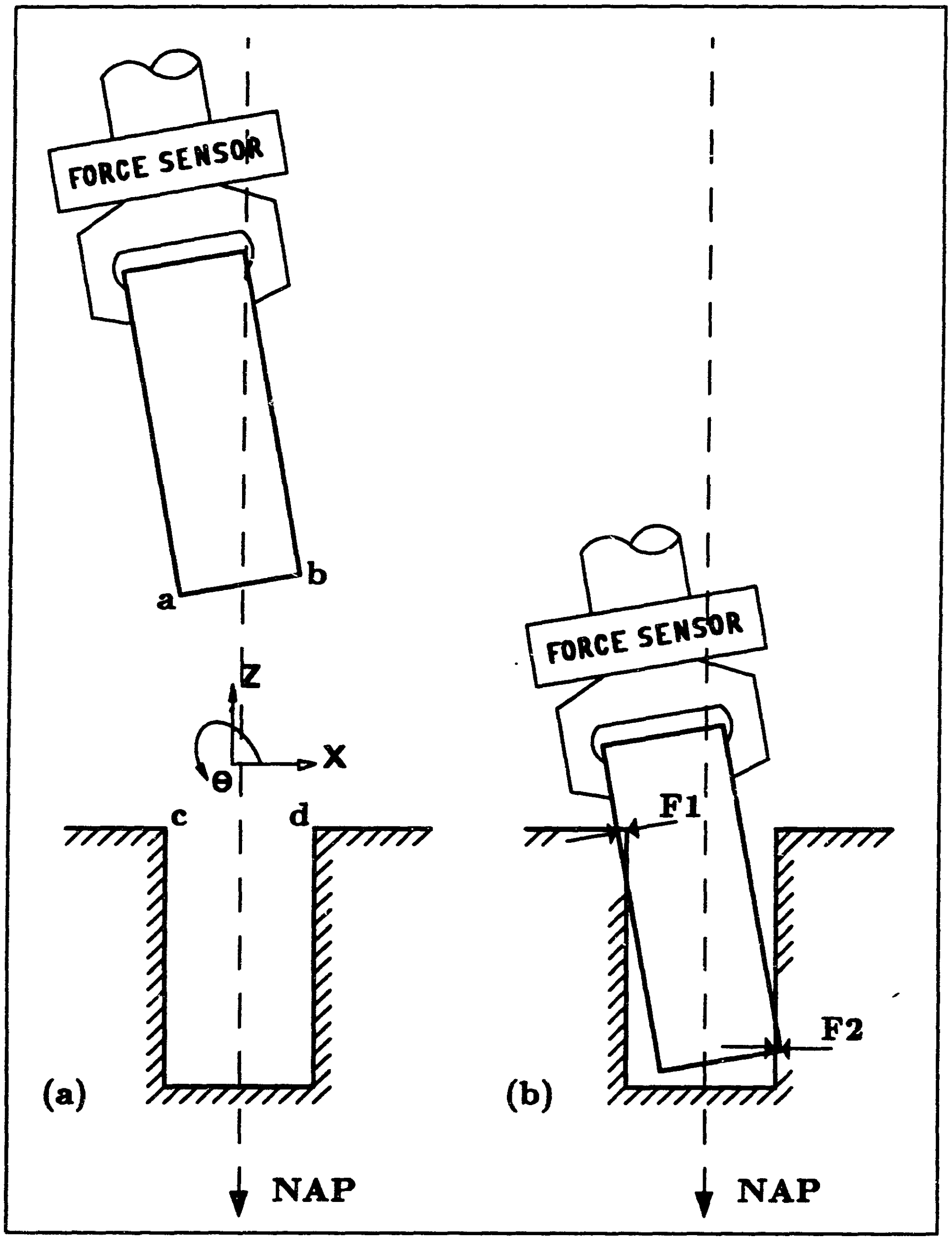

Figure C.1: The Peg and Hole Geometry for the Computer Simulation

commanded move and the distribution. The width of the distribution varies with both the normal force and the coefficient of friction. I was comfortable with this model right up to the point where we began running tests on the real system. The model is probably good for some robots, but not for the MITPAR, at least in its present configuration. With some regularity, controller overshoot and friction resulted in the system settling well *beyond* the commanded position.

C.3 Choice of Computer Hardware and Language

Common LISP was used because of the anticipated need for manipulating variable length lists of data. This proved to be a good choice. The functions that used the peg and hole geometries to generate the contact forces ended up looking a lot like FORTRAN, but once the program was running, manipulating data was very easy. The software development was done on a Model 3600 Symbolics Lisp Machine. The simulation runs were done on a LISP Machine, a VAX 750, or a Sun 3/75, depending primarily on the workload placed on the machines by other members of my research group.

C.4 Comparison of Simulated and Real System Behavior

The behavior of the assembly algorithm in the computer simulation and on the real system was very similar. There were some surprises, however. Most of them

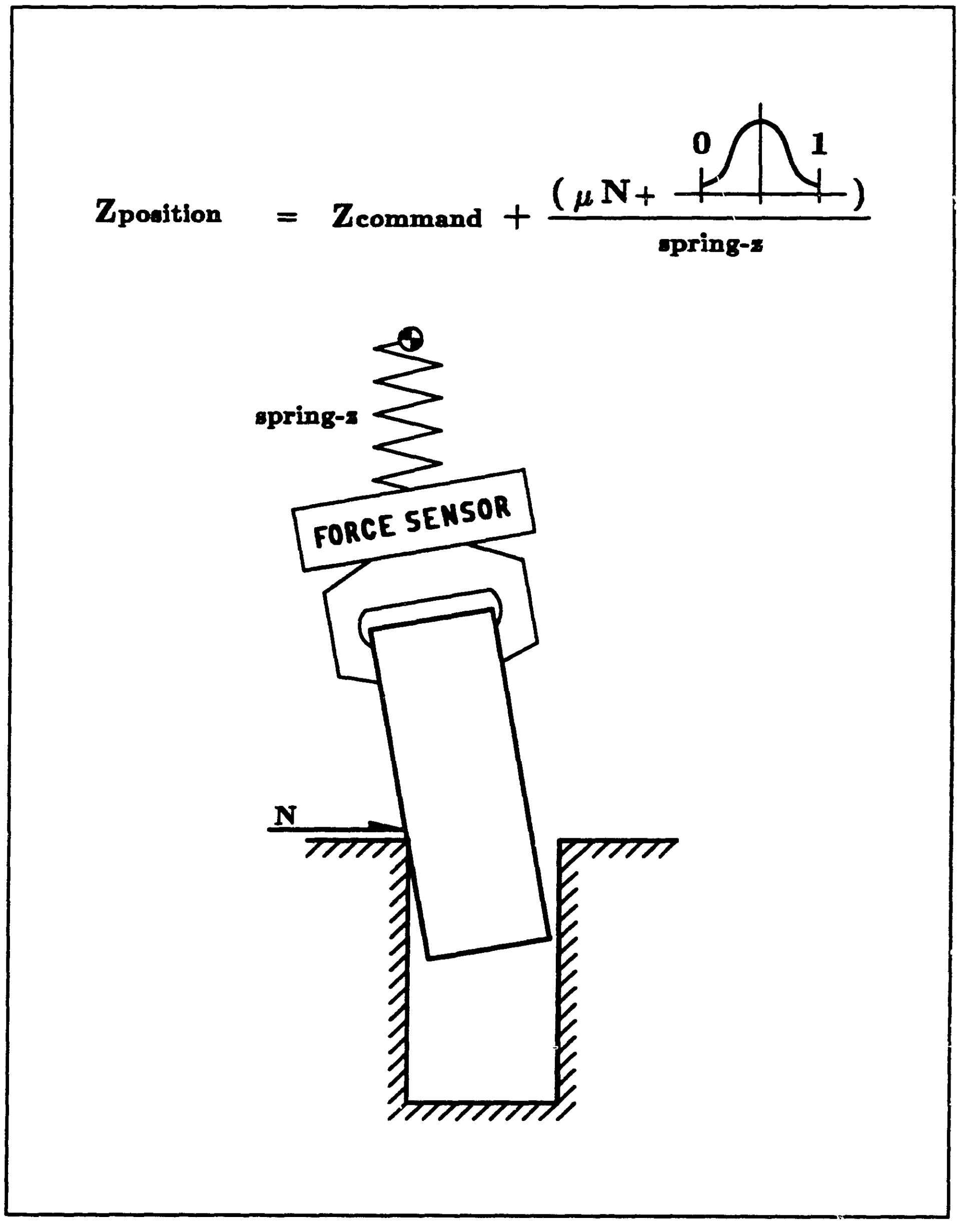

Figure C.2: A Simple Stochastic Friction Model

were due to errors in the computer model. In the simulation, the bottom of the peg was positioned over the center of the hole (to the precision of the computer), but the real system was set up by eye. This caused the real system to reach states that could not be reached by the computer simulation. After discovering this problem I added a function to the assembly process that introduced random errors in the initial X position of the peg (relative to the hole) that could be as large as one half the clearance between the peg and the hole. The real system typically reached about 50% more states than the simulation did as a result of this change. Convergence was slowed correspondingly.

The big error in the simulation (which fortunately had little effect on the behavior of the learning algorithm) was neglecting the effects of overshoot in the servos. I developed a simple stochastic friction model with the (naive) assumption that friction would only impede progress toward a goal position. In fact, if the system has any tendency to overshoot, friction can cause the final resting position of the axis to be beyond the setpoint. For most of the assembly cases there was relatively little information in the Fz and $\Delta F_z/\Delta Z$ states in the simulation even with what turned out to be an unreasonably conservative model of friction. On the real system, overshoot, friction, and the small angles of contact reduced the useful resolution of the F_z and $\Delta F_z/\Delta Z$ states from 2 in the simulation to 0 in the real system for many of the assemblies. The F_z and $\Delta F_z/\Delta Z$ states contained useful information only when the peg came close to jamming. These were very low clearance ratio (≈ 0.01) assemblies using large initial errors (≈ 5 degrees).

Appendix D

Assembly Algorithm Code

Much of the code that was written for this thesis is either implementation dependent, embarrassing, or both. In particular, there are many patches in the code that were necessary to get around timing problems caused by the many different ways that the different components (Section C.3) almost, but didn't quite, meet their published specs. The code also suffers the usual problems of developmental code - paths to nowhere, unused functions, and violated abstraction barriers. For the sake of clarity and brevity I am therefore including in this appendix only the functions at the core of the learning algorithm. These functions are MOVE_DRIVER(), BEST_MOVE(), and INSERT_NEW_DATA().

The MOVE_DRIVER() function contains the logic branching structure. It corresponds to the Performance Element defined in [Smith, et. al. 77]. The starred comment lines are used to point out the correspondence between parts of the code and the block diagram in Figure 2.3.

```
int
```

```
move_driver()
{
/* wait between moves until the system has settled ( the forces are */
/* within +/-0.05 lbs for more than one cycle of the lowest natural */
/* frequency of the system) */

  while((move_time_counter > move_rate) || settled) {

/******************(block 4, Flow Chart)*************************/

    /* check to see if we have hit bottom */
    if (z_grip < (Z_FINISH_POSITION +  LARGEST_EXPECTED_Z_ERROR)
&& (fabs((double)z_force) > fabs((double)Z_FINISH_FORCE))) {
      done = YES;
      success = YES;
      assembly_number++;
      insert_new_assembly_data(data_for_this_assembly);
      printf("This was a successful assembly\n");
/******************(block 7, Flow Chart)*************************/
      break;
    }

/******************(block 5, Flow Chart)*************************/
    if (forces_ok()) {


/* do nothing because no corrective moves were made during the last */
/* call to move_driver()  */

      if (x_move_direction == 0);

/* x_move_direction == 1 means that the corrective move made */
/* in the first X direction tried was successful. More than  */
/* corrective move is allowed, so the direction is multiplied */
/* by the number of moves made */

      else if (x_move_direction == 1)  {
data_for_this_assembly[visited_state_counter].move
  = (float)first_x_direction * x1_moves;
visited_state_counter++;
sum_of_x_moves += first_x_direction * x1_moves;

x_move_direction = 0;
t_move_direction = 0;
total_x_moves += x1_moves;
total_t_moves += t_moves;
x1_moves = 0;
x2_moves = 0;
t_moves = 0;

      }
/* x_move_direction == 2 means that the corrective move made */
/* in the first X direction tried was unsuccessful. */


      else if (x_move_direction == 2)  {
data_for_this_assembly[visited_state_counter].move
  = (float)(-first_x_direction * x2_moves);
visited_state_counter++;
sum_of_x_moves += -first_x_direction * x2_moves;
x_move_direction = 0;
t_move_direction = 0;
total_x_moves += (2 * x1_moves + x2_moves);
total_t_moves += t_moves;
x2_moves = 0;
x2_moves = 0;
t_moves = 0;

      }
      else
printf("something is wrong with the move_driver");

/* for calculating the force derivatives. The data from the force */
/* sensor is sampled several times after each move and then averaged */

      old_x_force = filtered_x_force;
      old_t_force = filtered_t_force;

/******************(block 2, Flow Chart)*************************/
```

```
      z_grip = z_grip - INCREMENT_IN_Z;

    }     indices

/* if the forces are not ok, a corrective move must be made. */
/* Different branches will be followed depending on the recent */
/* behavior of the system. */

/* If the forces got too high during the last move in Z */
      if (x_move_direction == 0 && t_move_direction == 0)  {

/* store the system position state so it can be recovered     */
/* if the forces go up with the first X move                  */

x_position_zero = x_grip;
t_position_zero = t_grip;

/* set random corrective move directions */

set_directions();

/*calculate the discritized values of the states */
set_indices();

/* store the state information */
/* (forces, force derivatives, and Z) */
data_for_this_assembly[visited_state_counter].index[0]
= x_force_index;
data_for_this_assembly[visited_state_counter].index[1]
= t_force_index;
data_for_this_assembly[visited_state_counter].index[2]
  = x_force_derivative_index;
data_for_this_assembly[visited_state_counter].index[3]
  = t_force_derivative_index;
data_for_this_assembly[visited_state_counter].index[4]
  = z_position_index;

/*****************(block 6, Flow Chart)***********************/

if (using_past_experience)
```

```
  first_x_direction = best_move();

/****************(block 3, Flow Chart)***********************/

x_grip = (x_grip + (first_x_direction * INCREMENT_IN_X));
x_move_direction = 1;
x1_moves++;
printf("1\n");
       }
       else if((x_move_direction == 1) && (t_move_direction == 0)
       && good_move) {
x_grip = (x_grip + (first_x_direction * INCREMENT_IN_X));
         x1_moves++;
printf("2\n");
       }
       else if((x_move_direction == 1) && (t_move_direction == 0)
       && (!good_move)) {
t_move_direction = 1;
t_grip = (t_grip + (t_direction_for_first_x * INCREMENT_IN_T));
x_grip = x_grip + 4 * t_direction_for_first_x * INCREMENT_IN_T;
t_moves++;
printf("3\n");
       }
       else if((x_move_direction == 1) && (t_move_direction == 1)
       && good_move)  {
t_grip = (t_grip + (t_direction_for_first_x * INCREMENT_IN_T));
x_grip = x_grip + 4 * t_direction_for_first_x * INCREMENT_IN_T;
t_moves++;
printf("4\n");
       }
       else if((x_move_direction == 1) && (t_move_direction == 1)
       && (!good_move)) {
t_move_direction = 2;
t_grip = (t_grip - (t_direction_for_first_x * INCREMENT_IN_T));
x_grip = x_grip - 4 * t_direction_for_first_x * INCREMENT_IN_T;
t_moves++;
printf("5\n");
       }
       else if((x_move_direction == 1) && (t_move_direction == 2)
       && good_move) {
```

```
t_grip = (t_grip - (t_direction_for_first_x * INCREMENT_IN_T));
x_grip = x_grip - 4 * t_direction_for_first_x * INCREMENT_IN_T;
t_moves++;
printf("6\n");
        }
        else if((x_move_direction == 1) && (t_move_direction == 2)
        && (!good_move)) {
x_move_direction = 2;
undo_moves = YES;
printf("7\n");
        }
        else if((x_move_direction == 2) && (t_move_direction == 0)
        && good_move) {
x_grip = (x_grip - (first_x_direction * INCREMENT_IN_X));
x2_moves++;
printf("8\n");
        }
        else if((x_move_direction == 2) && (t_move_direction == 0)
        && (!good_move)) {
t_move_direction = 1;
t_grip = (t_grip + (t_direction_for_second_x * INCREMENT_IN_T));
x_grip = x_grip + 4 * t_direction_for_second_x * INCREMENT_IN_T;
t_moves++;
printf("9\n");
        }
        else if((x_move_direction == 2) && (t_move_direction == 1)
        && good_move) {
t_grip = (t_grip + (t_direction_for_second_x * INCREMENT_IN_T));
x_grip = x_grip + 4 * t_direction_for_second_x * INCREMENT_IN_T;
t_moves++;
printf("10\n");
        }
        else if((x_move_direction == 2) && (t_move_direction == 1)
        && (!good_move)) {
t_move_direction = 2;
t_grip = (t_grip - (t_direction_for_second_x * INCREMENT_IN_T));
x_grip = x_grip + 4 * t_direction_for_second_x * INCREMENT_IN_T;
t_moves++;
printf("11\n");
        }
```

```
      else if((x_move_direction == 2) && (t_move_direction == 2)
      && good_move) {
t_grip = (t_grip - (t_direction_for_second_x * INCREMENT_IN_T));
x_grip = x_grip - 4 * t_direction_for_second_x * INCREMENT_IN_T;
t_moves++;
printf("to\n");
      }
      else if((x_move_direction == 2) && (t_move_direction == 2)
      && (!good_move)) {
failures = failures + 1;
printf("The assembly failed\n");
success = NO;
done = YES;
      }
    }

    /* Set the new positions to servo to */

    servo_to[0] = x_grip + relative_coordinates[0];
    servo_to[2] = z_grip + relative_coordinates[2];
    servo_to[5] = t_grip + relative_coordinates[5];

  }

}
```

The function BEST_MOVE simply sorts through the tist of visited states to see if the state the system is presently in has been visited before. If it has, the direction that has been the most successful in previous assemblies is returned. If the state is being visited for the first time, the function generates a random response.

```
int
best_move()
{
  int i, j, k;
  float sum;

  /* While looking at states that have been visited, compare the   */
  /* indices. If they all match, add up all of the previous moves. */
  /* If none of the states match, roll the dice...                 */

  i = 0;

  while (data_array[i].number_of_visits != 0) {

   if((data_array[i].index[0] == x_force_index)
       && (data_array[i].index[1] == t_force_index)
       && (data_array[i].index[2] == x_force_derivative_index)
       && (data_array[i].index[3] == t_force_derivative_index)
       && (data_array[i].index[4] == z_position_index)) {

      /* differentiating between number_of_visits is not really */
      /* be necessary until BOLD moves are attempted.           */

      printf("T"); /* make the terminal beep */
      for(k = 0; k < 500000; k++); /* pause */
      printf("T"); /* make the terminal beep */
      printf("***********visited this state before************\n");
      total_states++;

      if(data_array[i].number_of_visits < NUMBER_OF_SAVED_MOVES) {
sum = 0.0;
for(j = 0; j < data_array[i].number_of_visits; j++)
  sum +=  (float)data_array[i].corrective_move[j];

if((sum / data_array[i].number_of_visits) >= 0.0)
  return(1);
else
  return(-1);
      }
      else {
sum = 0.0;
for(j = 0; j < NUMBER_OF_SAVED_MOVES; j++)
  sum += data_array[i].corrective_move[j];
if((sum / data_array[i].number_of_visits) >= 0.0)
  return(1);
else
  return(-1);
```

```
      }
    }
    i++;
  }
  printf("T"); /* make the terminal beep */
  printf("***this is a new state***\n");
  new_states++;
  total_states++;
  return(random_plus_or_minus_one());
}
```

The function INSERT_NEW_DATA goes through the data list after each successful assembly and adds the new assembly data to the list. If a state was visited for the first time, it's indices are added to the end of the list.

```
int
insert_new_data()
{
  int i, j, k, n;
  int match;
  int this_move;

    printf("number of branch points = %d\n",visited_state_counter);

  for (i = 0; i < visited_state_counter; i++) {
    this_move = (int)data_for_this_assembly[i].move;

    /* step through all of the previously visited states     */
    /* to see if we have been there before                   */

    match = NO;
    j = 0;

    while(data_array[j].number_of_visits != 0) {
      match = YES;

      n = 0;
      /* step through the indices to see if they all  match.  */
      while (match && n < NUMBER_OF_INDICES) {
if (data_for_this_assembly[i].index[n]
    != data_array[j].index[n])
  match = NO;

n++;
      }

      /* If all of the indices match, insert the data into the    */
      /* data_array . Otherwise, continue until the loop gets     */
      /* to a state that has not been visited yet                 */
      /* before inserting the data                                */

      if (match) {
/* replace the oldest piece of data with the new data */

k = data_array[j].number_of_visits++;
data_array[j].corrective_move[(k % NUMBER_OF_SAVED_MOVES)]
  = sum_of_x_moves;
sum_of_x_moves -= this_move;
break;

      }
      else {
j++;
if (j == MAX_VISITED_STATES)
  printf("out of visited states\n");
      }
    }
    if (!match) {

      /* If the program gets here, there was no match of indices      */
      /* ( visits = 0). The new state is  then added to the data_array */
      /* printf("no match for i = %d\n", i); */

      new_visited_states++;
      for (k = 0; k < NUMBER_OF_INDICES; k++)
data_array[j].index[k]
  = data_for_this_assembly[i].index[k];
```

```
      data_array[j].corrective_move[0] = sum_of_x_moves;
      sum_of_x_moves -= this_move;
/*    printf("sum of directions for no match = %d\n", sum_of_x_moves); */
      data_array[j].number_of_visits = 1;

    }
  }

}
```