ARCHIMEDES: A System that Plans and Executes Mechanical Assemblies [1]

David Strip
Sandia National Laboratories
Albuquerque, N. M.

SAND--90-2400C

DE91 000036

Abstract

Archimedes is a prototype mechanical assembly system which generates and executes robot assembly programs from a CAD model input. The system addresses the unrealized potential for flexibility in robotic mechanical assembly applications by automating the programming task. Input is a solid model of the finished assembly. Parts relationships and geometric constraints are deduced from the solid model. A rule-based planner generates a "generic" assembly plan that satisfies the geometric constraints, as well as other constraints embodied in the rules. A retargetable plan compiler converts the generic plan into code specific to an application environment. Execution of the compiled plan in a workcell containing an Adept Two robot, a vision system, and other parts handling equipment will be shown on videotape.

I. Introduction

Archimedes is a prototype system for automating mechanical assembly. It accepts a solid model of an assembly as input and generates a program for executing the assembly.

Although robots offer the potential for truly flexible manufacturing systems, their application has usually been limited to situations in which the robot can operate in a relatively fixed manner for a reasonably long production run. The complexity of programming robots, designing fixtures, arranging workcells, and interacting with the output of the design process are the principle reasons for this limitation. The ***Archimedes*** system is motivated by the desire to significantly simplify the task of programming and setting up a mechanical assembly task. (We contrast mechanical assembly with electronic (PCB) assembly which is generally a 2 degree of freedom task and is well enough understood that commercial software is available.)

Our goal was a system which would accept a set of plans and parts as inputs and produce an executable assembly plan. We have elected to use solid models as input since they are increasingly available as the output of modern design systems and eliminate the need to build a solid model from multiple view drawings. When actual production is the goal, the executable assembly plan consists of instructions in the native language of the assembly robot. Our demonstration system, a workcell built around an Adept Two robot and associated equipment, reflects this goal. Live demonstrations take a solid model as input and produce a plan that is executed on the workcell. Other applications and the nature of their plans will be discussed later.

Automatic planning or simplified programming for robots has been a focus of robotics research from its inception. Early efforts in automating assembly include Fahlman's BUILD System [2], which focused on planning, but contains a very interesting approach to stability analysis. Several task level programming systems have been proposed, such as Lama [7] and AML/X [12], both of which use the idea of skeleton strategies. These are both textually oriented and procedural in nature. Rapt [11] moves away from the procedural format, and uses a textual input describing the geometric relationships among the parts. These task-level systems are primarily conceptual designs; little experience is reported on their application to assembly tasks using physical robots. Somewhat more closely related to our work are approaches that use a geometric representation as part of their input. Autopass [5] is an early example in this direction. Again, little is reported on the actual implementation or application of these concepts. To our knowledge, Lozano-Perez's Handey system [8] is the only other planning system that we are familiar with that emphasizes implementation. Related work in a different vein includes de Mello's [6] ideas on AND-OR graphs for finding feasible subassembly and related planning issues.

Recently there has been a flurry of activity in the mechanical assembly planning domain [9, 13, 4, 3, 1]. Most of this work is concurrent with the work we present here, and shares a common approach. In the following sections we will present an outline of our approach, and compare and contrast it with this body of work that is developing elsewhere.

II. A Motivating Example

Sandia National Laboratories acts as a design agency for the Department of Energy, designing electronic and mechanical components for safety and security. As a consequence, we are involved in the manufacture of complex, highly reliable mechanical assemblies which are built in very small lots. The stringent controls on assembly motivate our interest in automating the process. For the experiment described in this paper, we have selected the the pattern wheel assembly (shown in Figure 1). The pattern-wheel assembly is just one subassembly of the dual stronglink, a safety component designed

[1]This work was supported by the U. S. Department of Energy under Contract DE-AC04-76DP00789.

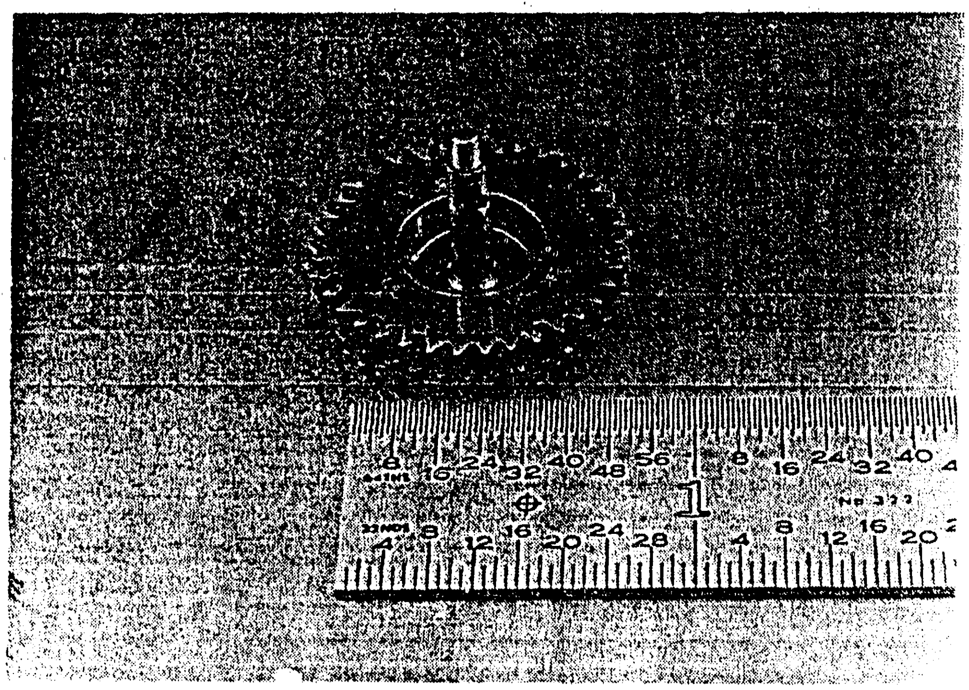

Figure 1: Pattern Wheel Assembly

at Sandia. This assembly contains 13 parts (mostly pins and cogs) and is representative of a large fraction of the approximately 150 parts in this clockwork-like mechanism. For assembly planning, the parts can be modeled as stepped cylinders with holes in them. This simple model is sufficient for all the parts in the pattern wheel assembly, as well as a substantial fraction of the other parts in the dual stronglink. The motions needed for this particular assembly are uni-directional, as are a great many of the motions needed for assembling other parts of the larger mechanism. This is consistent with the frequently cited study of Nevins and Whitney [10] on assembly. The goal of the experiment is to accept a CAD model of the assembled pattern wheel and the parts as input and to produce an assembled pattern wheel as output.

The workcell for assembling this component consists of an Adept Two manipulator with a tool changer, a two-finger pneumatic gripper, a vacuum gripper, a remote center compliance, a force sensor, an arm mounted video camera, and a VMEbus-based computing system to supplement the commercial robot controller.

III. The Input Model

Because solid models contain all the geometrical information about an assembly (at least in principle), they have become popular as the input to assembly planners. Since solid models increasingly form the underlying structure of modern design systems, this is a good choice if it can be made to work. Our approach makes few assumptions beyond the existence of the geometric data. Unlike [1], we avoid the use of features, since features are very function dependent. We would like, of course, to have assembly features identified to simplify our task. However, the machinist wants machining features, and so on. In addition, what constitutes a feature changes over time as well, as assembly or machining processes change. Unlike [4], we do not require relationships (*e.g.*, against or contacts) between parts to be expressly contained in the model. Instead, like [9, 1], we attempt to deduce them from the geometry.

Subassemblies are important in assembly planning because they limit the number of parts which must be considered at any stage by the planner, reducing the combinatorial growth of problem complexity. In addition, grouping parts into subassemblies can considerably simplify planning by revealing groups of parts that can be handled as units. These factors lie behind the interest in subassembly identification (*e.g.*, [6]). In examining the drawings for the entire stronglink, as well as other assemblies, it is our experience that the designer specifies subassemblies, often for inspection purposes in intermediate stages. There are sufficient levels of subassemblies in the drawings to keep the complexity in hand without identifying additional levels of subassemblies.

In our initial experiments we used a very simple CSG (constructive solid geometry) modeler that we developed for representing the domain of the pattern wheel parts (disks and pins which can be assembled by z-translation). It is implemented in Common Lisp on a Symbolics Lisp Machine. It can quickly answer the necessary queries about object intersection, interference in z-translation, and other similar queries required to infer part relationships. As restrictive as the assumptions may appear for the modeler used in the initial experiments, an examination of the service manual for the author's car revealed that large portions of the drivetrain, braking system, steering, suspension, and so on could be modeled and planned in this restrictive domain. We are currently increasing the scope of the ***Archimedes*** system, and have replaced the modeler with a more general 2 3/4 dimensional boundary representation solid modeler. The new modeler can accept input from AutoCad (.dxf format) files. In addition, the modeler has a well-defined interface to the planner, and can be replaced by any modeler capable of answering the defined geometric queries. The modeler operates in a server-client mode with the planner, and is free to run on any platform that supports IP-TCP connection.

The first stage in generating the plan is to create a partial order graph representing the geometric constraints on part order. We do this by querying the solid modeler about freedom of motion in the z-direction between pairs of parts. (This initial experiment uses an assumption of unidirectional assembly.) We then find the kernel of the pairwise relations, which gives us the minimum (or most constrained) representation of the part ordering. By generating candidate orders from the partial ordering, we restrict candidate sequences to always be geometrically feasible. One way to represent these constraints (with a loss of some of the information, however) is as an exploded diagram. (Exploded drawings are currently made by artists or by the operators of CAD systems, not automatically.) Figure 2 is an automatically generated exploded view. Using the solid modeler, the constraints among the parts are determined. The drawing is made by placing each part at the lowest possible level in the drawing such that it is above all parts that constrain it from below.

Modeling parts in a general 3D domain requires the use of a true 3D solid modeler. Many such modelers exist, and their design is a research topic in itself. Many of the tests for part interference that we now use can be couched as queries to a general solid modeler, although they take much longer to evaluate than in specialized modelers. The uni-directional assumption of this example cannot be expected to hold in general. For the 3D case, we are looking at a number of approaches for finding candidate directions for motion, both algorithmic, based on the part geometry, and knowledge-based, using information such as part categories or rules relating to previously discovered assembly directions.

IV. The planner

Given a candidate assembly order, the planner's main functions are to determine whether the candidate order is feasible, what assembly operations are required to move from one stage of the assembly to the next, and, if the order is infeasible, to select a next candidate order. Figure 3 shows an overview of these functions. The basic approach we have taken to the planner is to provide a set of rules for carrying out these operations. In the initial experiments the rules were implemented as functions in Common Lisp. We are modifying the system so that the rules are represented as predicates in the Joshua expert system substrate, which is a layered product for the Symbolics Lisp Machine. We are also implementing a more powerful backtracking mechanism in the modified system to provide more capabilities in searching the solution space.

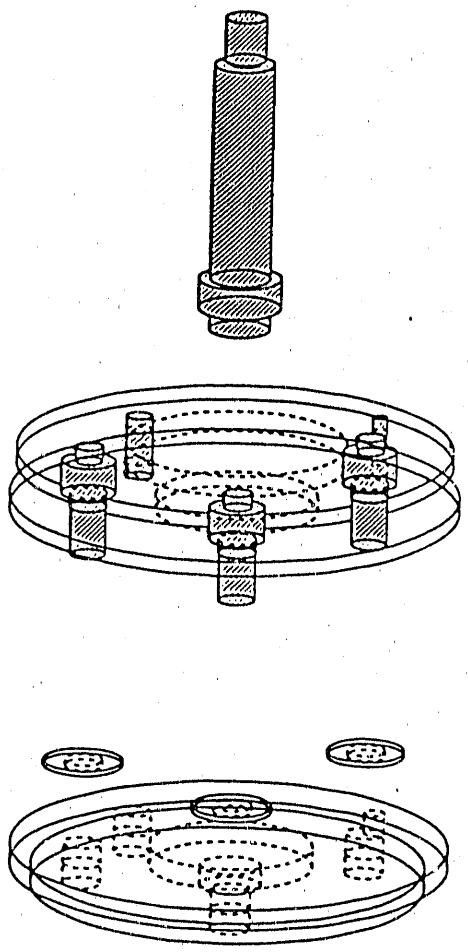
Figure 2: Automatically generated exploded diagram of pattern wheel assembly.

The planner currently recognizes three types of infeasibilities: instability of intermediate stages, inaccessibility of weld surfaces, and placing objects through the support surface. There are two cases for the instability check — objects that can fall over, and objects that may shift with respect to one another. The toppling instability rule operates on the basis of ratio of height to part diameter, a crude but effective measure in the part domain of interest. It would be a relatively simple matter to replace this with an actual measurement of the force required to move the center of gravity outside of the footprint, but that would be overkill in the domain of parts that can be manipulated by our workcell. The second instability check was provided to account for critical alignments required when welding two of the disk-like objects together. Note that in the exploded diagram the bottom object drawn consists of two disks. These are a subassembly which has been welded together. There is a requirement for a critical alignment between these parts. If no pins are through the holes in the two disks at the time of the weld, there is no assurance that the alignment is maintained. A rule in the planner reflects this situation, requiring pins in holes of all aligned disks. Thus, the two disks as

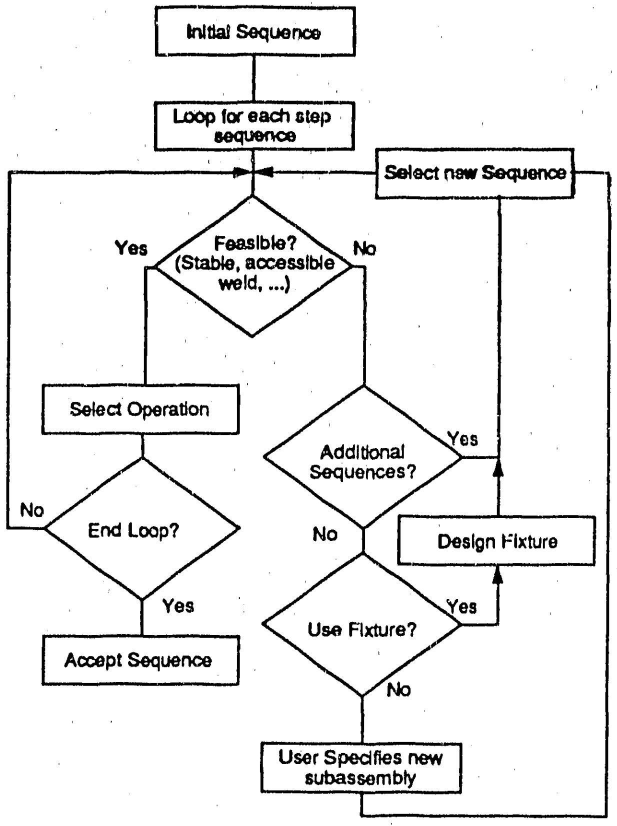

Figure 3: Oversview of Planner Functions

shown in the exploded diagram are considered unstable until either welded or pins are placed through them. The next rule covers accessibility of weld surfaces. The system is designed to use a laser spot-beam welder, held in the manipulator. The check is a straightforward visibility test. The final infeasibility check is for parts placed through the support surface. This occurs, for example, when a disk is placed on the table and a pin needs to be placed in a hole in the disk, and the bottom of the pin is below the level of the base of the disk. This check is purely geometric and is handled by the modeler.

For each candidate sequence, which is guaranteed to be geometrically feasible by the way in which we generate the sequences, the planner evaluates the addition of each part to the current assembly against the applicable rules. The planner continues as long as feasibility is maintained. When an infeasibility is uncovered, the planner applies a set of rules that derives a new sequence from the old. These rules either invert the sequence (effectively putting the assembly together upside ›ʋwn), put parts in as early as possible, or as late as possib.›. If reordering is not sufficient, the planner calls a fixture planning module that is capable of designing fixtures to handle the instability and part-through-support-surface infeasibilities. Fixtures are less desirable than re-ordered assembly sequences so they are called after reordering has been exhausted. At present the system has no provision for inverting portions of assemblies or discovering new subassemblies. In the current domain, rules for discovering these cases would be straightforward. (Mainly they need to recognize that all the currently assembled parts are rigidly connected and can be manipulated as a unit.) We provide a final "out" by allowing the planner to query the operator for a breakdown of the current assembly into two subassemblies.

As the sequence is evaluated for feasibility, another set of rules is applied as each part is added to the sequence to determine what operation is required to add the new part to the current assembly. The system has six operators: place, insert, multiple-insert, onsert, multiple-onsert, and weld. (Onsert is the opposite of a peg-in-hole operation — a hole-on-peg operation, such as placing a washer on a bolt.) There is no inversion operator, although we have mentioned that the system can put the part together "upside-down." This is not required because the generated plan contains orientation information which is used by the compiler to be discussed next.

At present, the planner contains no rules to measure plan quality — it merely looks for a feasible plan. Optimization is an important consideration, and will be addressed in extensions to the system.

The output of the planner is a generic plan — a sequence of invocations of the six primitive operators mentioned above. The plan contains calls to these operators which have parameters which are generally the parts involved, although other parameters, such as the specific weld surfaces sometimes are required. The order of calls in the plan is the assembly order. ˜ne plan also contains grouping information, showing which operations can be carried out in parallel, or are required to be serial. This information can be used to perform some optimization in the compiler.

V. The compiler

The compiler translates a generic, high-level plan into detailed instructions that can be directly executed in a robot workcell. (Other compilation targets will be discussed later.) In order to perform this translation, the compiler requires the geometric model of the parts, a map of the workspace, a skeleton for translating each primitive into robot specific commands. (Another view of this last part is that we have a macro for each assembly primitive, and the compiler expands the macro calls made in the plan.)

Because the planner only deals with the sequencing and operation selection, the compiler is required to plan the workspace and maintain a record of its status. The workspace manager allocates and maintains a record of the reachable workspace of the robot for which the plan is being compiled. Parts are assigned absolute positions within the workspace based on their bounding spheres and order of use. The strategy seeks a compromise between minimizing workspace fragmentation (which would dictate a global best fit policy) and minimizing manipulator motion based on the proximity of related parts. The workspace manager is also able to reclaim portions of the workspace to be reused later in the assembly. This space is most commonly used to store subassemblies that have been created from the individual parts which had occupied this area. Because subassemblies may be added to the next level assembly in an orientation inverted relative to the orientation in which they were assembled, the workspace manager is required to keep track of all part orientations. The compiler is responsible for planning ancillary actions required for inverting parts; the generic plan contains no information on these motions.

While the workspace manager is responsible for assigning the position of a part, the nominal orientation of a part is specified by the grasp planning module. The philosophy of the grasp planner is to choose the initial orientation at which a part is presented to the workcell in such a way that regrasping is unnecessary. The grasp planner generates an initial set of grasps based on the no regrasp criterion. The resulting set of grasps is then mapped to the part's starting position and pared on the basis of stability. The set of orientation trajectories resulting from pairing each valid starting configuration with the set of valid ending grasps is then compared to the kinematic capabilities of the robot being used in the assembly. This procedure identifies all of the initial part orientations which result in a physically achievable motion trajectory that does not require regrasping. A unique solution from this set can be obtained by using the stability ranking, a dexterity measure, or a combination of the two.

The outputs of the workspace manager together with the grasp planner specify a nominal workcell design. However, due to the positional uncertainty inherent in material handling systems, the actual position and orientation of the parts presented to the workcell will vary. To accurately identify the locations of these parts at assembly time the compiler generates code with calls to a vision system. Since the nominal position of a part is already known, the system does not need to deal with part identification except to flag errors. The majority of this calculation is performed off-line at compile time. A single matrix multiply is all that is required at assembly time in order to perform a least-squares fit of the data to the part model. The advantages of this approach are faster execution time and flexibility in the target hardware.

After establishing absolute workcell locations and grasps for all parts in the assembly, the compiler translates the planner language commands into the control language for a specific system controller. The compilation is performed by merging the location information with the parameters to assembly primitives and expanding the skeletons for the assembly operations. For the Adept workcell, the compiler produces V+ code which is downloaded to the Adept controller. This downloaded program controls all robot motions and auxiliary device control and communicates with the vision system over the controller serial link.

The use of the skeletons and macro expansion-like compilation allows us to easily retarget the compilation to different robots and applications. By compiling code for various workcells, we view the system as a manufacturing tool. Other compilation targets, however, change our perspective on the system's use. We developed a set of skeletons which target GSL code for the IGRIP computer graphic simulation system. Initially this target was used in debugging the compiler code. The simulation target has a much more important use in the product design phase. By allowing easy access to simulation of the actual assembly plan, designers can quickly and accurately evaluate the impact of design changes on manufacturability. Thus, we can view ***Archimedes*** as a design analysis tool. Yet another compilation target might focus on the time required for each operation. By applying this set of expansions to a complex system such as a ship, which might take years to build, we can produce PERT diagrams of the assembly, allowing estimation of material requirement dates and identification of critical paths in the assembly. In this light, ***Archimedes*** is a production planning tool.

VI. Experimental Results

The planner and compiler described here were initially applied to the pattern wheel assembly presented earlier. We first compiled the plan into GSL code for the IGRIP simulation system and executed the assembly as a simulation. As noted, this was of great assistance in debugging the code generator. When the workcell was completed, we began compiling into Adept's V+ programming language. The primary difficulties that arose in this stage were similar to those encountered when manually programming a robotic task: accurately defining fixed locations in the workcell, getting various pieces of equipment to talk to one another, etc. Instead of using the compiler generated locations, we used a manually designed part tray that would allow part to be grouped in kits. The vision system was used to locate the large cog-like parts, while the small pins were precisely located in holes in the part kit delivery tray. This corresponds to a situation in which the cogs are provided by an outside manufacturer and are manually added to the kits, while the pins are fabricated in-house and are precisely located in the part tray as they leave the turning station. The force sensor in the initial system is only used for guarded moves (and searches) rather than force-directed actions. A passive RCC on the arm provided the required compliance for the tight insertions. In spite of the limited capabilities of the workcell, the compiler generated code allows consistently successful assembly of the pattern wheel assembly. The plan creation and compilation each take less than one second to execute (on a Symbolics Lisp Machine and a Sun workstation, respectively). The assembly requires a few minutes in the workcell.

The ***Archimedes*** system is capable of dealing with a large variety of interesting real-world assemblies, including a parts of the automobile drivetrain such as the transmission and differential; many subassemblies in machine tools; many household appliances like blenders, mixers, drills, etc; consumer electronics like VCRs and personal stereos ("Walkman"); and so on. We are taking advantage of the modular implementation of the system to expand its capabilities. We are extending the modeler to more general 2 3/4 D capabilities, and working on a true 3D approach. We are improving the planner with a more powerful backtrack mechanism and a simple method for writing rules. The compiler will have added capabilities for motion planning and error handling.

References

[1] A. DelChambre, "A Pragmatic Approach to Computer-Aided Assembly Planning," *Proc. IEEE Int'l Conf. on Robotics and Automation*, Cincinnati, OH, 1990.

[2] S. E. Fahlman, "A Planning System for Robot Construction Tasks," Ph.D. Thesis, M. I. T., Cambridge, MA., 1973.

[3] R. Hoffman, "Automated Assembly Planning for B-Rep Products," *Proc. IEEE Int'l Conf. on Systems Engineering*, Pittsburgh, PA, 1990.

[4] H. Ko and K. Lee, "Automatic Assembling Procedure Generation from Mating Conditions," *Computer Aided Design*, Vol. 19, No. 1, Jan./Feb., 1987.

[5] L. I. Lieberman and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," *IBM J. Research and Development*, Vol. 21, no. 4, July, 1977.

[6] L. S. Homem de Mello and A. C. Sanderson, "And/Or Graph Representation of Assembly Plans," *Proc. AAAI 86*, Philadelphia, PA, 1986.

[7] T. Lozano-Perez and P. Winston, "LAMA: A Language for Automatic Mechanical Assembly," *Proc. 5th Int'l Joint Conf. Artificial Intelligence*, Cambridge, MA., 1977.

[8] T. Lozano-Perez, *et. al.*, "Handey: A Robot System that Recognizes, Plans, and Manipulates," Proc. IEEE Int'l Conf. on Robotics and Automation, Raleigh, NC., 1987.

[9] R. S. Mattikali, P. K. Khosla, and Y. Xu, "Subassembly Identification and Motion Generation for Assembly: A Geometric Approach", *Proc. IEEE Int'l Conf. on Systems Engineering*, Pittsburgh, PA, 1990.

[10] J. L. Nevins and D. E. Whitney, "Assembly Research," *Automatica*, Vol. 16, 1980, pp. 595–613.

[11] R. J. Popplestone, A. P. Ambler, and I. Bellos, "An Interpreter for a Language Describing Assemblies," *Artificial Intelligence*, Vol. 14, No. 1, pp 79–107.

[12] R. H. Taylor, P. D. Summers, and J. M. Meyer, "AML: A Manufacturing Language," *Int'l. J. Robotics Research*, Vol. 1, No. 3, Fall, 1982.

[13] J. D. Wolter, "On the Automatic Generation of Plans for Mechanical Assembly," Ph.D. Thesis, University of Michigan, Ann Arbor, MI., 1988.

Acknowledgements

The author would like to acknowledge the contribution of Tony Maciejewksi, who designed and wrote the bulk of the plan compiler described here. I would like to thank Cliff Loucks and Colin Selleck for making the robotic systems work.