To be presented at the International Symposium on Robotics and Automation, July 18, 1990, Vancouver, BC.

ARCHIMEDES: AN EXPERIMENT IN AUTOMATING MECHANICAL ASSEMBLY

DAVID STRIP
Sandia National Laboratories
Albuquerque, N. M.

ANTHONY A. MACIEJEWSKI
Department of Electrical Engineering
Purdue University
West Lafayette, IN.

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. Using this model, ***Archimedes*** deduces geometric assembly constraints and then produces an assembly plan that satisfies the geometric constraints, as well as other constraints such as stability and accessibility. A retargetable plan compiler converts the generic plan into robot and cell specific code, including recognition routines for a vision system. In the prototype system the code is executed in a workcell containing an Adept Two robot, a vision system, and other parts handling equipment.

1 Introduction

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

Our goal is a system which accepts a set of plans and parts as inputs and produces an assembled product. In more practical terms, this translates into providing a solid model as input to a computer program and producing as output the necessary manipulation and sensing programs for carrying out the assembly on a robot and ancillary systems. As we are concerned with manufacturing systems, we assume that the environment can be be structured in order to simplify the task, although we must obviously restrict the degree of structure in order to maintain our goal of flexibility.

This work was performed at Sandia National Laboratories and supported by the U. S. Department of Energy under contract DE-AC04-76DP00789.

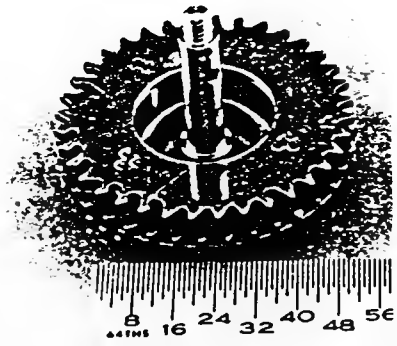

Figure 1: Pattern Wheel Assembly

Early efforts in automating assembly include Fahlman's BUILD System [1], which focused on planning, but contains a very interesting approach to stability analysis. Several task level programming systems have been proposed, such as Lama [4] and AML/X [8], both of which use the idea of skeleton strategies. These are both textually oriented and procedural in nature. Rapt [7] 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 [2] 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 [5] is the only other planning system that we are familiar with that emphasizes implementation. Related work in a different vein includes de Mello's [3] ideas on AND-OR graphs for finding feasible subassembly and related planning issues. In our work we assume that important information on the breakdown into subassemblies is part of the design information, although provision is made for identifying additional subassemblies that are important for assembly purposes.

2 Overview of the *Archimedes I* Experiment

Figure 1 show the pattern wheel assembly from the dual stronglink, a safety component designed at Sandia National Laboratories. This assembly contains 13 parts (mostly pins and cogs) and is representative of a large fraction of the over 250 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 [6] on assembly. The goal of the ***Archimedes I*** 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 controller.

The modeling and planning software, written in Common Lisp, are executed on a Symbolics Lisp machine. The plan compiler, which takes the output of the planner and produces code for the robot controller, is written in C++ and executed on a Sun 3. The VMEbus system contains several processors; some are specialized for image processing, while others are single-board 68020 systems programmed in C++ and operating under VxWorks. The Adept controller is connected to the VMEbus system via a serial interface. Our concept is to develop a robot program offline and then download the program to a commercial controller for execution, using the robot controller to handle the (anticipated) error conditions and take the appropriate branching action. In line with this concept, the Adept program is generated offline on a Sun and downloaded to the controller via a serial link.

3 The modeler

The role of the solid modeler in ***Archimedes*** is to represent parts in the assembly and to answer the types of queries necessary to generate an assembly plan. Most available solid modelers are unable to respond to our queries in a reasonable amount of time. In addition, most are unable to represent assemblies and variational information such as tolerances. Fortunately, the relative simplicity of the part domain makes it possible to write a very fast, domain-specific modeler to address these issues.

The modeler we developed is able to model stepped cylinders with arbitrarily located stepped holes, although all cylinder and hole axes must be parallel. This makes queries such as detection of interference (null object detection of the intersection of two components) computationally straightforward. Collisions of two objects on certain simple paths are also very easy to detect in this domain. While this initially sounds like a very restrictive parts domain, examination of the service manual for a car of one of the authors reveals that the bulk of the drivetrain, as well as many other sub-assemblies of the car could be modeled in this domain.

In order to generate the set of allowable assembly sequences, we need to first find which parts interfere with others when moved in given directions. Here the uni-directional nature of the assembly speeds the process by reducing the number of interference checks that we need to make. The 2 1/2 dimensional nature of this domain allows the interference test to be reduced to a series of simple planar tests, which can be carried out quickly. Applying this operation to the pattern wheel assembly we can find the constraints which parts place on one another. 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.

Moving from this simple domain to a more general domain will occur in several stages. We are now working on a generalized 2 1/2D modeler. This will retain much of the speed of the existing system while enriching the domain to include

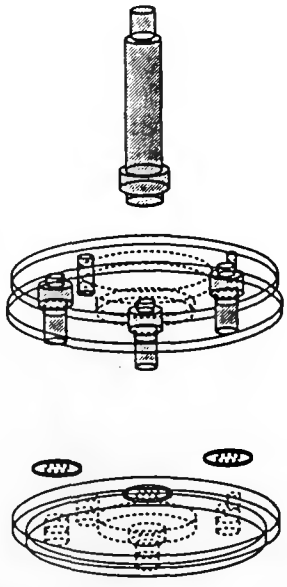

Figure 2: Automatically generated exploded diagram of pattern wheel assembly.

many currently manufactured designs. 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 may take longer to evaluate than in the specialized modeler. 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.

4 The planner

There are many possible orders in which the parts may be assembled based solely on the geometric constraints determined by the solid modeler. Some of these orders may be inefficient; for example, many tool changes are required. Others may be infeasible due to instability of intermediate stages. The role of the planner is to evaluate candidate orders, determine their quality, and try to find a good order. Since we view the context of our problem as the batch manufacturing environment, planning is really a one-time offline task, so there is a strong motivation to spend the time to optimize the plan, which is to be executed many times.

The planner currently recognizes 4 types of infeasibilities: instability of intermediate stages, inaccessibility of weld surfaces, placing objects through the support surface, and out-of-order assembly. The planner takes a candidate sequence and evaluates it for feasibility against these four criteria. (In fact, the system is arranged to guarantee that only valid orderings are generated, so the last criterion does not have to be checked.)

For each type of infeasibility, there are one or more fixes that the planner can try. For instability, the alternatives are a different sequence or a fixture. Inaccessible

surfaces may be made accessible by inverting parts or subassemblies. Fixtures or resequencing are used when parts penetrate the support surface. The search among the various possible assembly orders is controlled by assigning a "score" to a sequence based on the type and number of infeasibilities, and backtracking when the an operation fails to improve the score. Currently, the search stops when a feasible solution is found; no attempt is made at further optimization.

It is possible that no feasible sequence can be found using these operators. This is because no allowance is made for inverting a partially completed sub-assembly. The system allows the definition of new sub-assemblies as a solution to this problem.

Some assemblies, including the pattern wheel assembly in our example, cannot be assembled by a single manipulator without the use of fixtures. Our planner is capable of designing certain simple fixtures when necessary to allow development of feasible plans. The fixture design is output as standard dimensioned 3-view drawings, familiar to any machinist.

5 The compiler

The compiler is the part of the system that translates a generic, high-level plan into detailed instructions that can be directly executed in a robot workcell. The central principle in the development of ***Archimedes*** is modular design to maintain flexibility. Fundamental to this concept is the separation of the compiler front-end, which deals with a workcell-independent specification of allowable assembly plans, from the back-end which is responsible for generating the machine-specific control commands for the robot(s) and ancillary systems. As with traditional computer language compilers, information is passed between the modules through the use of a symbol table.

Input to the compiler is generated by the planner, and is in the form of an assembly plan which consists of a sequence of calls to assembly primitives. This assembly plan, together with the solid model of the completed assembly, specifies all of the *relative* motions between individual parts required to complete the assembly. One of the primary responsibilities of the compiler is to bind these relative part positions to absolute workcell locations. This mapping is performed by a workspace manager module which is a three-dimensional analog of the run-time memory management found in computer systems.

The workspace manager is responsible for maintaining the status 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.

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 the 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 for all parts in the assembly, the compiler translates the planner language commands into the control language for a specific system controller. The compiler is currently capable of outputting V code to run the Adept manipulator and GSL code for the IGRIP computer graphic simulation system, which was used in debugging the compiler code. The ability to easily modify the compiler for alternate manipulator controllers is a direct result of the separation between the assembly-dependent responsibilities of the compiler from the robot-dependent responsibilities.

6 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 kitting. 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 kitted 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 exectute. The assembly requires a few minutes in the workcell.

The ***Archimedes I*** 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 1/2 D capabilities, and working on a true 3D approach. We are also adding sophistication to the planner, and adding capabilities for motion planning and error handling.

References

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

[2] 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.

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

[4] T. Lozano-Perez and P. Winston, "LAMA: A language for automatic mechanical assembly," *Proc. 5th Int'l Joint Conf. Artificial Intelligence*, Cambridge, MA., 1977.

[5] 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.

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

[7] 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.

[8] 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.

Acknowledgements

The authors would like to thank Cliff Loucks and Colin Selleck for their assistance in making the robotic systems work.