SANDIA REPORT

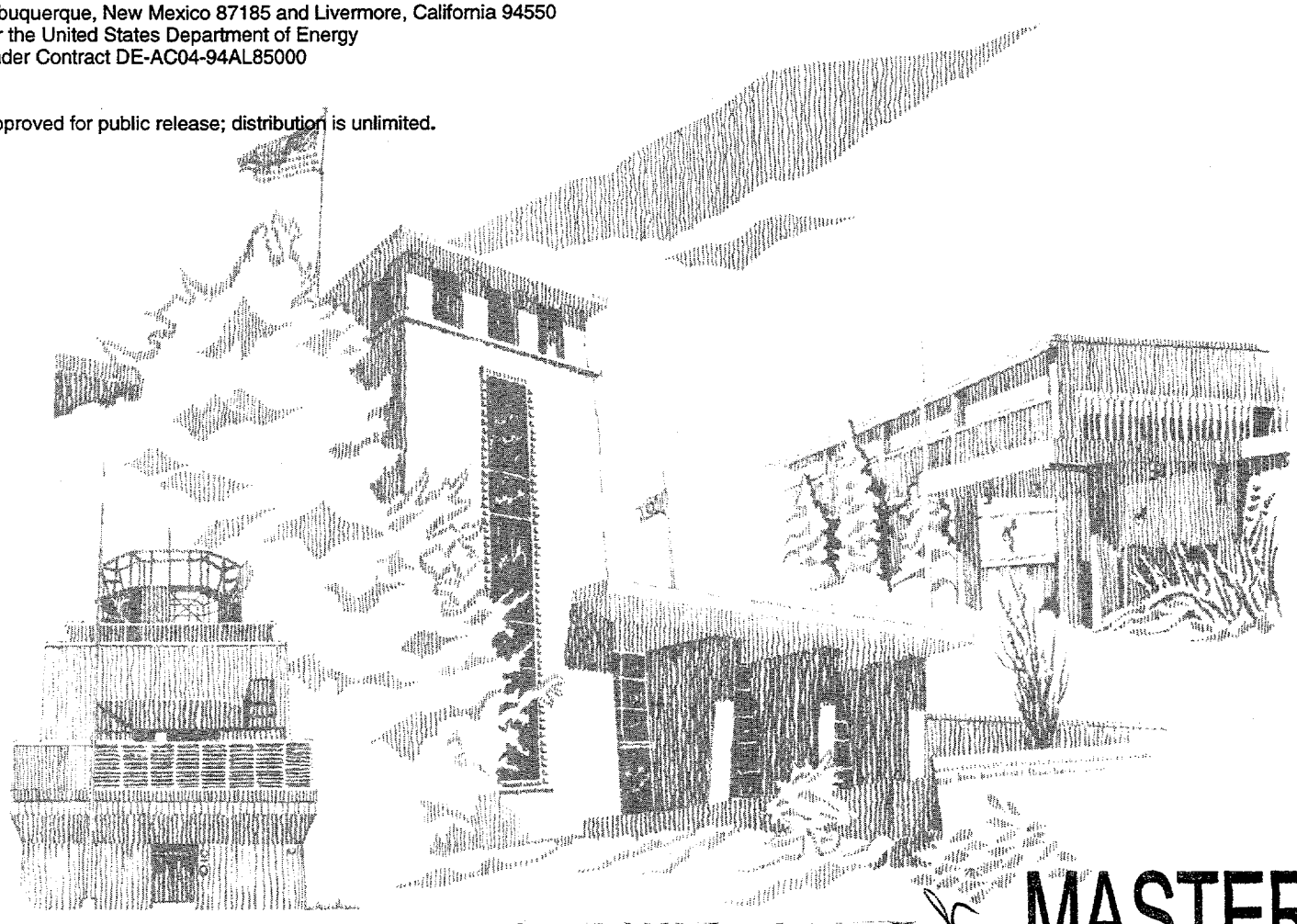
SAND96–0433 • UC–705
Unlimited Release
Printed November 1996

LDRD Final Report: Automated Planning and Programming of Assembly of Fully 3D Mechanisms

Stephen G. Kaufman, Randall H. Wilson, Rondall E. Jones, Terri L. Calton, Arlo L. Ames

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

Approved for public release; distribution is unlimited.



Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND96-0433
Unlimited Release
Printed November 1996

Distribution
Category UC-705

LDRD Final Report: Automated Planning and Programming of Assembly of Fully 3D Mechanisms

Stephen G. Kaufman Randall H. Wilson Rondall E. Jones
Terri L. Calton Arlo L. Ames

Intelligent Systems and Robotics Center
Sandia National Laboratories
Albuquerque, New Mexico 87185

Abstract

This report describes the results of assembly planning research under the "Automated Planning and Programming of Assembly of Fully 3D Mechanisms" LDRD. The *assembly planning problem* is that of finding a sequence of assembly operations, starting from individual parts, that will result in complete assembly of a device specified as a CAD model. The *automated assembly programming problem* is that of automatically producing a robot program that will carry out a given assembly sequence. Given solutions to both of these problems, it is possible to automatically program a robot to assemble a mechanical device given as a CAD data file. This report describes the current state of our solutions to both of these problems, and a software system called Archimedes 2 we have constructed to automate these solutions. Because Archimedes 2 can input CAD data in several standard formats, we have been able to test it on a number of industrial assembly models more complex than any before attempted by automated assembly planning systems, some having over 100 parts. A complete path from a CAD model to an automatically generated robot program for assembling the device represented by the CAD model has also been demonstrated.

Contents

1 Introduction and Motivations

A complex mechanical device often requires an even more complex system to assemble it. When many identical devices are to be assembled, the effort and capital required to design and fabricate an automated assembly system are well-repaid. In an environment of small-lot production, however, such as that characterized by DOE weapons components, such costs cannot be amortized over long production runs. Therefore the devices are often assembled by hand. Robotic technology has made it possible to perform many different assembly tasks with the same machine, but the costs of tooling and programming the robot for each new task, as in the case of hard automation, can be too high. But robots do offer the possibility of more precise and repeatable assembly than is possible by hand. It is therefore possible that automatic methods of programming robots for assembly tasks could significantly reduce the production costs, and increase the quality, of mechanical assemblies.

Before a robot can be programmed to perform an assembly, the assembly plan must be known. Given the increasing complexity of electro-mechanical devices, automatic methods of finding such plans can be of great use to product and manufacturing system designers. They can enable design for assembly, by providing analysis of the assemblability of a design as expressed by a CAD model. Such methods have the possibility to significantly reduce the costs of product realization, whether or not robots are intended to be used in the final assembly system. Output from such a system can be instructions to a human assembler, an animation of the assembly sequence, or an animation of an assembly system carrying out the assembly. It is only incidental that the sequence can also be translated into code driving a robot controller.

The former problem, the *automated assembly programming* problem, is an instance of the notoriously difficult problem of task-level robot programming ([23], [22]). Robot programs typically must specify all details of the desired manipulation, including tool poses, speeds and accelerations, grasping and ungrasping actions, and so on. Robot programming would be much easier if only the tasks to be performed need to be specified, and the robot fills in the details that are now given by the human programmer. We do not make a serious attempt to solve this problem in any generality.

Instead, we concentrate on the latter problem of *assembly planning*. It is the problem of taking a CAD model of a mechanical device and computing one or more assembly sequences that will assemble the device from its piece parts. This is actually a wide class of problems, depending on the constraints the resulting sequence must satisfy. The minimum constraint is that the sequence must be geometrically valid — that is, none of the operations of joining parts and subassemblies require interpenetration of parts. Another constraint is that any tools that are required to perform assembly operations must be able to go through a sufficient range of motion to fulfill their function. We may also require that the available assembly operations be limited to those that can be performed by a given assembly apparatus. This makes possible an automatic assessment of the assemblability of a product on an existing, or proposed, assembly system.

Another version of the problem is to find many, or all, possible assembly sequences,

possibly scored according to a user-defined cost function.

Because of the importance of the assembly planning problem, the last decade has seen an explosion in computer-aided assembly planning research (see, e.g., [12]). Significant advances have been made in both theory and practice, and a great number of experimental systems have been built. The research performed under this Laboratory Directed Research and Development program (hereafter, "LDRD") addresses many of these problems, and in some areas defines the state of the art. In particular, the system we have developed plans sequences for real assemblies from industry and the Defense Programs sector, some containing over 100 parts. These assemblies are much larger than those attacked by other planning systems. We have also demonstrated a complete and automatic path from a CAD model of a weapon component, to an optimized assembly sequence, to an automatically programmed robot that carries out the sequence. To our knowledge, no other research program has demonstrated this completeness. We have also developed animation facilities, both for showing an assembly as "flying parts", and as a simulated workcell performing an assembly.

The rest of this report is organized as follows. We first describe the system we have constructed. The system is called "Archimedes 2", because it is a successor to the prototype Archimedes system described in [30], also developed at Sandia. The inputs and outputs of the system are described, and then its structure. Then the results of experiments (assemblies planned and executed) are described. This is followed by a section discussing what we learned in building the system, and justifications for some of the major design decisions. The report closes with future technical work planned by our team, and plans for deploying the technology. There is no section describing related work; instead, such work is referenced at appropriate places throughout the paper.

2 Overview of the Current Assembly Planning System

Archimedes 2 includes two assembly planners. One considers only part geometry, and finds a single plan, if one exists. The actual planning architecture is very simple, and its power derives almost exclusively from the large suite of geometric functions and queries we have developed. Its success demonstrates the power of this suite of geometric functions, and we therefore call this planner the "geometric engine".

The second planner is built on top of the geometric engine, using a standard search algorithm that simultaneously considers many assembly sequences. It also considers the orientation of the current subassembly and the grippers needed to acquire and place the parts, preferring sequences that minimize inversion operations and gripper changes. Currently, it does not consider tool accessibility.

Both of these planners require geometric models of the complete assembly as input. Additional information about joining (e.g., welds and pressfits) is also necessary, and (for the geometric engine) directions of assembly. Both produce output suitable for producing animations, either in "flying parts" form or a simulation of a complete assembly workcell performing the assembly. The optimizing planner can also produce output suitable for

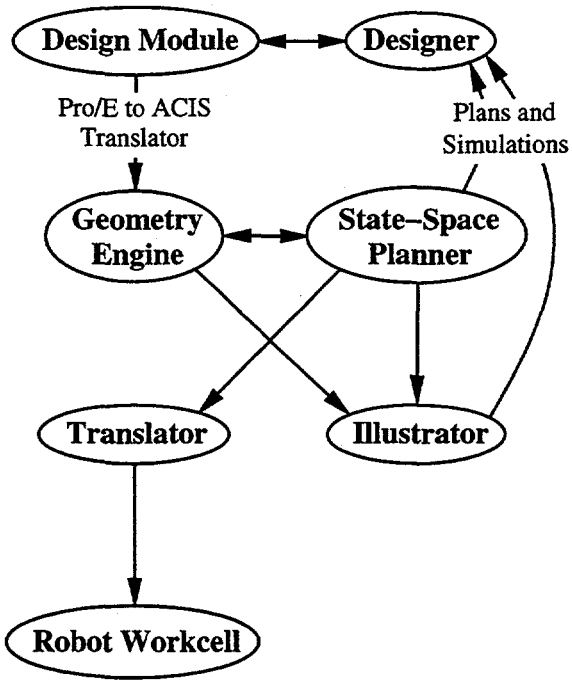


Figure 1: The architecture of Archimedes 2

translation into robot code.

Besides the two planners, the Archimedes 2 software consists of an animation facility ("Illustrator"), a translator of high-level plans into robot control code, and a library of robot control subroutines. The system also includes an Adept2 robot as the target of this translation. Related to Archimedes 2 is a "design module", based on Pro/ENGINEER®, that provides CAD data. It is not intrinsically a part of Archimedes 2, but we will describe it briefly to give an idea of how our input data can be obtained.

The relationships among the components are illustrated in figure 1.

We will first describe the assumptions we made in developing the software, and then the inputs and outputs to the Archimedes 2 system as a whole. Then, each system component will be described in greater detail.

2.1 Background and Assumptions

The plans produced by our planners have the following properties. No more than two "hands" are needed at any time, where a hand is a gripper or a fixture. Also, when a part is placed into a subassembly, its relative position with respect to the other parts of the subassembly does not change. That is, a part is moved only once by itself, and thereafter only as part of a subassembly. In the jargon of the assembly planning literature, we say

that the sequences are two-handed and monotone. Furthermore, subassemblies as well as piece parts may be moved in placement operations.

Our software uses the ACIS®solid modeling system to represent the geometry and topology of parts and subassemblies of assemblies. The following is a brief overview of ACIS; an understanding of it (or at least some of the terminology) is essential to understanding some of the later material.

ACIS is a boundary-representation solid modeler. Boundary surface types include plane, sphere, cylinder, cone, torus, and NURB spline surfaces. Physical objects are represented as *bodies*, which may contain one or more *lumps*, each of which corresponds to a single physical object. Lumps have *faces*, which are the bounding surfaces; the bounds of the surfaces are called *loops*. Loops are taken from a large collection of curve types; the class of representable shapes is extremely large.

ACIS also has a facetting capability, which we use in an essential way. Each body is facetted during model input; the facetted representation is used for contact finding, collision detection, and animation.

Each part in an assembly is represented by a data structure that includes an ACIS body, and a facetted version of the body.

Archimedes 2 encodes the following four major assumptions:

- All parts are perfectly rigid
- The available assembly operations are

  mating which puts a part in its correct relative position (with respect to the other parts in the assembly). This operation requires specifying the pair of parts or subassemblies to be mated, and a trajectory for one of them to follow.

  joining which attaches two parts once they are in the correct relative position. These include welding, screwing, glueing, etc.

  inverting a part or a subassembly.
- Mating trajectories are translations, rotations, or a combination.
- All operations are reversible, so that we may compute a disassembly sequence, and reverse it to obtain an assembly sequence.

2.2 System Input

Input to the Archimedes planners is of two types: solid models of assemblies (in their assembled configurations); and additional information about how parts are joined, recommended subassemblies, and suggested assembly directions. These types of inputs are gathered together into an *assembly file*. An assembly file contains a list of filenames describing parts or subassemblies; associated with each is a 4×4 transformation matrix that specifies the pose of the part or assembly in the containing coordinate system. (The containing coordinate system for the top-level assembly is the world coordinate system; the containing coordinate

system for a subassembly is the coordinate system of its containing assembly). Files describing parts are ACIS-readable models; files describing subassemblies have the same form as the top-level assembly file.

Each subassembly has an associated file describing non-geometric data. Such non-geometric data is a standard feature of assembly planning systems. Bourjault's *liaisons* [4] and Ko and Lee's *mating conditions* [17] are early examples, while Homem de Mello's *relational model* [13] is probably the most comprehensive view. Work on CAD standards such as PDES/STEP [15, 21] promises to standardize representations for much of this information.

Specifically, the auxiliary files declare:

- threaded contacts, pressfits, and snapfits. The geometric queries will correctly report part overlap for such matings, leading to the inference that the matings cannot be realized. Yet the matings are indeed realizable, via screwing motions and part deformation, respectively. The planners simply assume that any such matings will be made successfully, and perform no geometric checking for them. The geometric engine requires that directions be specified for such matings.

- Recommended subassemblies.

- Tools needed to perform certain assembly operations, so that their accessibility can be checked. Due to lack of space, we cannot describe that work in the present paper; see [32] for a description.

An example assembly non-geometric data file is shown in figure 2; it corresponds to the subassembly in figure 5 consisting of the two toothed wheels and the five pins joining them. This assembly is called a *pattern wheel*. The line numbers in the first column do not appear in the file itself; they are included for easy reference in the text of this report. Any line beginning with a pound sign is a comment, and is ignored by the parser. As shown, the file is not complete; weld specifications for two of the five pins have been left out for brevity. The interpretation of each line will be described.

Line 1 This line declares the units to be inches. This is relevant for lines 8 through 16.

Lines 2 through 7 These lines declare part contacts to be interference fits. So, as described above, the geometric queries will report that the matings cannot be performed.

The last three integers of lines 2 through 7 declare the direction in which the first named part must approach the second named part. This information is used by the geometric engine, not the optimizing planner. The three digits specify a unit vector, in world coordinates, corresponding to a translation trajectory.

Lines 8 through 16 These lines declare the tool needed to perform the part mating operation, the point of application of the tool, and the direction of its application.

A final note: a part may appear several times in an assembly. The part `alpin` is an example, as three instances of it occur in this subassembly. Pressfit declarations naming such

```
#  .anf file for pattern-wheels and alignment pins of pattern wheel assembly

1. units inch 1.0

2. pressfit alpin a3 0 0 1
3. pressfit unpin  a3 0 0 1
4. pressfit hipin   a3 0 0 1
5. pressfit alpin a2 0 0 -1
6. pressfit unpin  a2 0 0 -1
7. pressfit hipin   a2 0 0 -1


#   2-part-tool toolname part1_NN part2_NN  x y z  dx dy dz

# the following weld sites, 3 per pin, have radius .0315
# one right, one up-left, one down-left, at 120-deg spacing

8. 2-part-tool laserspot1 alpin_02 a2_01 .3365 0 0  0 0 1
9. 2-part-tool laserspot1 alpin_02 a2_01 .2893 .0273 0  0 0 1
10. 2-part-tool laserspot1 alpin_02 a2_01 .2893 -.0273 0  0 0 1

11. 2-part-tool laserspot1 alpin_03 a2_01 -.2465 .101 0 0 0 1
12. 2-part-tool laserspot1 alpin_03 a2_01 -.2937 .1283 0 0 0 1
13. 2-part-tool laserspot1 alpin_03 a2_01 -.2937 .0737 0 0 0 1

14. 2-part-tool laserspot1 alpin_04 a2_01 .0315 -.305 0 0 0 1
15. 2-part-tool laserspot1 alpin_04 a2_01 -.0157 -.2777 0 0 0 1
16. 2-part-tool laserspot1 alpin_04 a2_01 -.0157 -.3323 0 0 0 1
```

Figure 2: Example of an assembly non-geometric data file

a part refer to *all* instances of contacts between the two named parts. In these examples, lines 2 and 5 refer to three pressfits each, since there are three instances of alpin and one instance each of a3 and a2.

The reader may wonder why some of the contacts must be identified by the user as input to the system. The answer has to do with finding the proper balance between effort on the part of the human and the machine. It is very easy for the human to recognize and specify these contacts, but very difficult to develop a feature-recognition algorithm that will do the same. CAD systems of the future will probably allow such specification of contact types. Our experience with data from Pro/ENGINEER, in which such contact information

is not available, is that it takes hours (not tens of hours) to find and specify such contacts. Compare this effort with the months that would be required to program the appropriate feature recognition functions, which would still fail in some cases.

2.3 System Output

The Archimedes 2 system has several forms of output. From the least to the greatest level of detail, they are:

1. Textual plans, consisting of high-level operations such as place, weld, and invert (see figure 3).
2. Animations of plans, showing parts being brought together without any tooling or fixturing (this is like an animated exploded diagram). These animations can also include inversion operations.
3. Animations of workcells performing the assembly, with robots, tooling, and fixturing.
4. Robot control code sufficient to control robotic assembly of the given device.

2.4 System Components

2.4.1 Design Module

Currently under development, the Archimedes design module will allow modeling the geometric and other aspects of a product necessary for planning. In the past, the auxiliary (non-geometric) information has for the most part been available on final drawings, but not in a computer-accessible form. Built on Pro/ENGINEER, the design module currently supports detailed weld and adhesive specifications, with part and assembly geometry handled by Pro/ENGINEER. Other part attachment information (such as snapfits, interference fits, and threaded attachments) will soon be supported. The design module can translate the geometric and auxiliary data into ACIS for input into the downstream Archimedes planners. In some cases, data from other ACIS-compatible CAD systems is used by entering the auxiliary data by hand.

2.4.2 Geometric Engine

The geometric engine is a rewriting of the methods of [33] in C++ using the ACIS geometric modeling kernel. The geometric planner determines sequences of assembly operations that are geometrically valid, i.e. those in which parts do not collide with each other. The planning method consists of three major computations, which are now described.

```
(mate hipin_06 with wheel_fixture by (0, 0, -1) using parallel_jaw)
(mate unpin_05 with wheel_fixture by (0, 0, -1) using parallel_jaw)
(mate alpin_02 with wheel_fixture by (0, 0, -1) using parallel_jaw)
(mate alpin_04 with wheel_fixture by (0, 0, -1) using parallel_jaw)
(mate alpin_03 with wheel_fixture by (0, 0, -1) using parallel_jaw)
(mate a2_01 with hipin_06 unpin_05 alpin_04 alpin_03 alpin_02
by (0, 0, -1) using vacuum)
(weld a2_01 to hipin_06 at 0.2281, -0.225, -0)
. . .
(invert SubAssemblySet: a2_01 alpin_02 alpin_03 alpin_04 unpin_05 hipin_06)
(mate a3_07 with hipin_06 unpin_05 alpin_04 alpin_03 alpin_02
by (0, 0, -1) using vacuum)
(weld a3_07 to hipin_06 at 0.2281, 0.225, 0.129)
. . .
(mate spacer_08 with a3_07 by (0, 0, -1) using parallel_jaw)
(mate spacer_09 with a3_07 by (0, 0, -1) using parallel_jaw)
(mate spacer_10 with a3_07 by (0, 0, -1) using parallel_jaw)
(mate 1lockring_11 with spacer_10 spacer_09 spacer_08 a3_07
alpin_04 alpin_03 alpin_02 by (0, 0, -1) using vacuum)
(mate shaft_13 with wheel_fixture by (0, 0, -1) using parallel_jaw)
(mate 1elgearo_12 with shaft_13 1lockring_11 alpin_04
alpin_03 alpin_02 by (0, 0, -1) using vacuum)
(weld 1elgearo_12 to shaft_13 at 0.0627, 0, 0.2133)
. . .
```

Figure 3: High level planner output. Many similar weld operations have been omitted.

Contact Finding Before planning begins, all part-part contacts are found by computing all face-face contacts. Input models include planar, spherical, cylindrical, conical, and toroidal faces, leading to 25 possible face contact types. Of these, we have implemented only four, which are given (with examples) in table 1. In addition, a *threaded* contact type is defined, which is used to model the contact between a screw and the threads into which it is turned. These five contact types have been sufficient to capture virtually all of the contacts found in the examples described below. The few unsupported types we have encountered have not affected the geometric correctness of the assembly sequences computed.

Each contact type has a specialized routine to determine if a contact of that type exists between two faces. The facetted representation is frequently used in these routines.

Trajectory Selection Part matings are used to select possible mating trajectories. For example, if two planar faces of distinct parts are in contact, the feasible infinitesimal trajectories of each part are restricted to those such that the scalar product of the trajectory

Contact Type	Examples
plane-plane	stacked plates
plane-cylinder	cylindrical bearing
plane-sphere	ball bearing
cylinder-cylinder	peg in hole
threaded	screws, bolts

Table 1: Contact types used in the Archimedes 2 Geometric Engine

vector and the outward-pointing contact normal of the moving part is non-positive. The allowable trajectories of a part, given the totality of its contacts with other parts, is the intersection of the sets of trajectories allowed by each contact. This information is stored in a structure called a *non-directional blocking graph*, or *NDBG* [35], which is used to compute feasible partitionings of the assembly into subassemblies, and to select trajectories for parts or subassemblies.

Trajectory Checking If a part can move at all, the distance it can move before colliding with any other part must be sufficient to remove it completely from the rest of the assembly. An obvious way to do this is with volume sweeping computations, such as those provided by ACIS. When parts have complex geometries, however, such methods are prohibitively slow. We therefore developed a new method of collision detection, which exploits the hardware capabilities of a graphics workstation. The method is raster-based, discretizing the silhouettes of the (facetted representation of) parts in a plane perpendicular to the direction of motion, but otherwise makes no simplifying assumptions or approximations.

A brief summary of the available graphics hardware is required to understand how the method works. The displayed image has color intensity values stored in a hardware buffer, called the image buffer, for each pixel. Another hardware buffer, called the z-buffer, stores the *distance* of each pixel from the viewpoint. This buffer is typically used in rendering 3D scenes in which occluded objects should not be displayed; it works as follows. Each pixel of the z-buffer is initialized to the maximum distance. For each object displayed, each pixel that will be in its displayed image is checked for its distance to the viewpoint. If it is less than the current distance stored in the z-buffer, the image buffer is updated with the intensity value of the pixel. If it is greater than the current distance, the object is occluded, and so the image buffer is not updated.

The graphics hardware allows the distance comparison to be a greater-than test instead of less-than. This setting can be modified from the program; it is the key to the fast checking. To test if part A will collide with part B when A is translated to infinity, the z-buffer is first cleared. The viewpoint is set to be the opposite of the sweep direction, so the sweep is effectively away from the viewpoint. Part A is drawn in black, then part B is drawn in white, with the greater-than distance comparison. Any pixels of part B that

are outside the boundary of part A will not be drawn, since they are not further than the maximum distance to which the z-buffer has been initialized. If any pixels of part B are drawn, it is because they are further than the pixels of part A; since we are viewing along the sweep direction, this means that part A will collide with part B at some point along an infinite translation. Therefore, we need only check for the existence of white pixels in the image buffer. This is easily done using the programming interface to the graphics system.

These operations are all performed in hardware. Translations of any direction can be tested, since the graphics hardware applies the proper transformations to the parts when drawn. We have observed that the planner runs nearly three orders of magnitude faster when using the hardware-based method than when using the volume sweeping method.

Planning with the Geometric Engine The geometric engine has a very simple planning strategy. Parts or subassemblies are tested for removability in turn; if one may be feasibly removed, the planner recurses on the remaining subassembly. When all parts have been removed, the recursion is unwound, resulting in a sequence of parts and trajectories.

2.4.3 Optimizing Planner

The optimizing planner considers many possible disassembly sequences, and finds a sequence of lowest cost according to a given cost function. The planner is based on an algorithm that searches a tree structure of partial assembly decompositions of the given assembly. Each path through the tree represents a different partial assembly sequence; at each search step, the current best partial sequence is decomposed further. This process repeats until the decomposition is complete. The search algorithm used is an implementation of the A* search algorithm [28], which is guaranteed to find a path of minimum cost under certain assumptions.

Input for the planner is the same as for the geometric engine, described in the previous section. Output is a list of actions required to perform the assembly.

We describe the optimizing planner as follows: assumptions it encodes; representation of a state; and planning strategy.

Assumptions The planner must respect the capabilities of the target workcell, so that it does not produce a plan that is impossible to execute. The optimizing planner makes four assumptions that correspond to workcell capabilities.

1. Subassemblies may be in one of two orientations, normal and inverted. One orientation is obtained from the other by rotating the subassembly 180 degrees about the x axis. We presume that there is a device in the workcell that can perform this inversion operation.

2. All insertion operations are translations in the downward (z) direction. This is a simplification which is consistent with typical usage of SCARA robots used in the Archimedes 2 workcell. The assumption covers a large class of assemblies, and is less onerous than it may appear to be.

3. There are sufficient grippers in the workcell to manipulate each part; the mapping between parts and the tools needed to manipulate them is specified to the planner.

4. All joining operations can be performed by a laser spot welder. Although our workcell does not have such a welder, it performs assembly as though it did; welding operations are simulated by replacing unwelded assemblies with welded assemblies.

States A state is a data structure representing the parts that have not yet been decomposed during the planning process, the current orientation of the subassembly consisting of these unremoved parts, and the number of tool changes and subassembly inversions needed to assemble the remaining parts (these two values are functions of the path to the state, not the state itself). The state also includes information about the part or subassembly last removed.

Search Strategy Search begins with the complete assembly. For each part not in a recommended subassembly, and each recommended subassembly, a state is generated corresponding to the removal of the part or subassembly. These states are tested for geometric feasibility, and for visibility of all weld sites to the laser welder (geometric feasibility testing is implemented by the geometric engine). If a state fails these tests, it is inverted, and the tests are reapplied. This is the only way inversions are introduced into the plan. If the tests fail again, the state is deleted.

If the proposed state passes these tests, it is deemed to be a successor to the given state. The number of tool changes and the number of inversions are updated, by comparing the tool needed for the removal to the the previously used tool, and comparing the orientation of the successor state to that of the given state. If either comparison fails, the respective count is incremented by one. The cost of the state is then computed as the sum of these two values, and the number of remaining parts.

When all successor states have been generated, the lowest cost state must be selected. To do this efficiently, a priority queue, or heap, is used to hold all states that have not yet had their successors generated. The front element of the queue is guaranteed to be a state of lowest cost.

Search terminates when a state with no unremoved parts is found, or when there are no states left to expand. The first case corresponds to success, the second to failure. In case of success, the the list of states from the final state to the start state is traced, and this list is transformed into the corresponding list of actions.

Recall from section 2.1 that there are three types of action: insertion of a part or subassembly, inversion of a subassembly, and welding. Each state transition implies an insertion operation, and may also require an inversion and a number of welds. The translation is straightforward; if the states joined by the transition have different orientations, an inversion precedes the insertion; if there are welds between the inserted part or subassembly and other parts, the appropriate weld operations follow the insertion.

When the plan for the assembly is complete, a plan is computed for each declared subassembly. This continues recursively until all subassemblies have been planned, or one

is encountered for which a plan cannot be found.

Note that when recommended subassemblies are used, the search becomes a hybrid approach, not a pure A* search. That is, the search is not over as large a space as would be required to assure optimality. The tradeoff between computational time and optimality seems unavoidable. In practical terms, guiding the planner by suggesting reasonable subassemblies seems workable.

2.4.4 Code Generator

The code generator, or translator, produces a sequence of V+ function calls that, when executed, assemble the pattern wheel (illustrated in figure 5). Inputs to the translator are the high level plan, a file describing the location of objects (such as fixtures) in the workcell, and a file describing grasp points of the parts.

The translation is straightforward, essentially a process of replacing high-level operations with calls to their counterparts in the V+ library. Parameters of these functions are computed, or read from files. We claim no novelty in addressing this problem of automatic robot programming; indeed, LAMA [24] and AUTOPASS [20] were more sophisticated. Like us, those systems had a collection of skeletal procedures, but the applicability of the procedures was checked by simulating the operation in a model of the workcell. In LAMA, the simulation results were used to predict the contacts, associated forces, and thence error conditions to check during the assembly process. AUTOPASS skeletal procedures had associated preconditions and postconditions. A procedure's preconditions are evaluated with respect to the model; if satisfied, it could be put into the evolving program. Then the procedure's postconditions were used to update the model of the assembly. Hence, unlike us, these systems added flesh to the skeleton. They also used motion planning to assure that the pick and place operations were collision-free, whereas we assume that the "skyhook" motion planning will avoid collisions.

2.4.5 Workcell

The target of the translation process is a workcell consisting of an Adept2 robot, two arm-mounted cameras, a force sensor, a quick-change wrist, a parallel jaw gripper with built-in RCC, a vacuum gripper, an inverter, and a parts kit. Three fixtures are bolted to the workcell floor for assembling the pattern wheel. The vision system and force sensor controller are both Adept products. A large collection of V+ routines was written for the workcell; calls to these routines are a large part of the output of the translation process. We first describe the workcell hardware, and then the software library.

Workcell Hardware We begin with a description of the parts kit, as it is the input to the assembly process. The kit is a tray with four quadrants, one for each of the four disks of the pattern wheel assembly. The disks are not in known positions within the quadrants, nor is it known in advance which disk is in which quadrant. The other parts of the assembly are in known locations around the edge of the tray. Asymmetric fiducial marks in the center of

the tray are used by the vision system to determine the tray's pose, from which the location of the pins and spacers may be computed. The vision system is also used to find the precise poses of the four disks.

The Adept2 is a small SCARA robot, with a quoted repeatability of 5 thousandths of an inch. By using Adept's High Accuracy Positioning System (HPS), which maps the kinematic errors of the robot, and corrects these errors when the robot is commanded to move to a position, we were able to obtain *positional* accuracies of 2 to 3 thousandths of an inch.

The two arm-mounted camera are used to find the poses of the parts tray and the four disks. Both are mounted about thirty inches above the floor of the workcell. One has a 25 mm lens and the other a 50 mm lens (with extension tubes to allow focussing on the floor), leading to fields of view of about 2 and 1 inches, respectively. The first camera finds approximate poses; the approximate positions are used to guide the second camera to obtain more precise positional data. The second phase was found to be necessary to achieve accuracies sufficient for the assembly to succeed.

The force sensor is used to implement guarded moves, which are used whenever a part is acquired or placed. Such moves terminate when the force sensor detects contact as it moves downward.

The vacuum gripper is used to grasp the disks. All other parts are grasped with the parallel jaw gripper. The latter gripper's built-in RCC, and the positional accuracy obtainable with the HPS data, suffice to assure the success of the pin insertion operations.

The inverter is a parallel jaw gripper, whose faces are parallel to the workcell floor. It is attached to an actuator that rotates the gripper 180 degrees, along an axis also parallel to the floor. The inverter is in turn attached to the floor. Parts and subassemblies are presented to the inverter with the vacuum gripper, which then backs off while the actuator inverts the grasped object. The vacuum gripper then re-attaches itself to the object, the inverter's jaws open, and the now-inverted object is returned to a fixture.

Workcell Software The workcell software library consists of four major classes of routines. These are: vision and parts recognition, workcell device control, utilities, and "task-level" functions. All are written in V+, Adept's proprietary language. We describe each in turn.

Recall that we do not address the problem of automatically programming object recognition routines. Therefore the recognition functions are all written by hand. Functions in this category implement the two-phase procedure for finding accurate positions of the tray and the four disks, and for calibrating the cameras with the robot's coordinate frame and the HPS data.

Workcell device control routines drive the actuators to cycle the inverter, open and close the gripper, lock and unlock the RCC, open and close the quick-change wrist, and so on. Utility functions include such staples as sorting.

Of greatest interest to our work are the "task-level" functions. These correspond more or less to the operations found in the high-level assembly plan, which are pick and place,

invert, and weld. Since our workcell does not have a welder, the weld operations become no-ops.

Pick and place tasks are implemented by two functions, one for the parallel-jaw gripper and the other for the vacuum gripper. The first has parameters for the pickup and placement locations and approach and depart heights. When executed, this routine causes the robot to move to the point at the given approach height above the pickup point, move downward by a guarded move to the pickup point, close the gripper, and move upward to the departure height. It then does a joint-interpolated motion to the point above the placement point, executes a guarded move downward, and releases the part.

The vacuum gripper pick and place function has an additional parameter, the grasp offset. Its action is similar to the previous function, but considers the grasp offset for the part so that the gripper is not put on a hole. It also considers the possibility that the part may need to be flipped, since the disks can be upside-down in the parts tray (this is recognized by the vision routines).

The function implementing part and subassembly inversion has parameters for offsets in x, y, and θ. These are used in presenting the part to the inverter, so that the gripper does not collide with the inverter. The part is moved to the position specified by the offsets (which are relative to the center of the inverter jaws), and the inversion cycle is begun.

The software library consists of about 180 kilobytes of V+ code. About half is used for vision and recognition routines. No checking for success of the operations is performed.

2.4.6 Illustrator

The Archimedes 2 Illustrator is a facility for viewing animations of assembly sequences carried out by a simulated robotic workcell. It is based on CimStation, a software product of SILMA, that is expressly designed for 3D graphical simulation of manufacturing systems. CimStation includes extensive tools for describing workcells, and can import data from Pro/ENGINEER to obtain part models. It is therefore well-suited for producing animations of robotic assemblies.

Illustrator Input The Illustrator was designed to allow animation of arbitrary workcells performing arbitrary assemblies. Therefore it requires as input an assembly sequence and workcell description. The latter can include

- geometrically and kinematically accurate robot models
- models of tooling, fixtures, and ancillary equipment (such as inverters)
- simulation control

The robot models are available from SILMA; models of other workcell parts can be imported from Pro/ENGINEER. These models are integrated with descriptions of all relevant poses of workcell components into a CimStation "cell" file. In addition, supplementary data about each of the grippers is encoded in a separate "workcell representation file" which

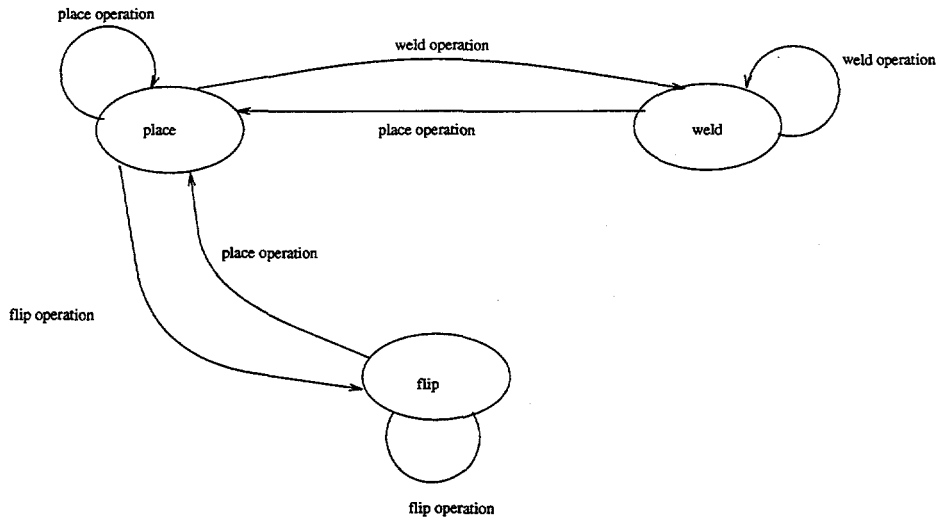


Figure 4: The finite-state machine implemented by the Illustrator

currently just provides the CimStation file name of each gripper to be referenced, and its basic specifications (its function is expected to be expanded later.) For example:

```
remark robot = adept

parallel_jaw file = ~/cim/projects/public/tools/cgripper.ee
parallel_jaw grip_minimum = .15 cm
parallel_jaw grip_maximum = 7.1 cm

bgripper file = ~/cim/projects/public/tools/smallgripper.ee
bgripper grip_minimum = 3 cm
bgripper grip_maximum = 7 cm

vacuum file = ~/cim/projects/public/tools/vacuum_blue.ee
vacuum grip_minimum = 0 cm   (this is irrelevant for vacuum grippers)
vacuum grip_maximum = 100 cm (this is irrelevant for vacuum grippers)
```

Simulation control is described as a finite-state machine, currently as program text in the SIL language used to program the CIMStation simulator.

Illustrator Operation The main stages in the simulation process are:

- The CAD model parts are installed into the simulated workcell, and the parts are assembled into a replica of the Pro/ENGINEER model by referring to the assembly

information generated by the Pro/ENGINEER to ACIS translator discussed above.

- The assembly plan from the Archimedes planner is read, looking for the order in which parts are referenced and the insertion direction used. The parts are moved from the assembly to a simulated "parts tray" in the order of use, and in an appropriate orientation. Parts not used in a particular plan are hidden from view.
- The assembly plan is then read a second time, and each action is processed by the finite state machine controller of the illustrator. The workcell can be in one of three states, called "place", "weld", and "flip". Figure 4 illustrates the finite-state machine implementing the control of the illustrator. Fragments of SIL code implementing this machine are shown below.

```
didit:=false
if not didit then
  repeat
    case cellstate of
      'assembly': doassystate(action); {action is an input line to perform}
      'weld':     doweldstate(action);
      'flip':     doflipstate(action);
      ...
    end;
  until didit
```

Here, each state is handled by a separate routine which must be able to react appropriately to every statement type. For example: the outline of the procedure that handles the "assembly" state is:

```
task doassystate(action:string);
  case verb of
    'place': {perform the action...};       didit:=true;
    'weld':  {transition to weld state...}; cellstate:='weld';
    'invert':{transition to flip state...}; cellstate:='flip';
     ...
    'quit':  {cleanup as required...}
  end;
```

3 Experimental Results

We have run the planners on a variety of examples; illustrations of three of the assemblies are shown in figures 5, 6, and 7. Table 2 summarizes planning times required by the two planners for various assemblies (figure 5 is the assembly of the second line of the table, figure 6 is the assembly of the seventh line, and figure 7 is the assembly of the fifth line).

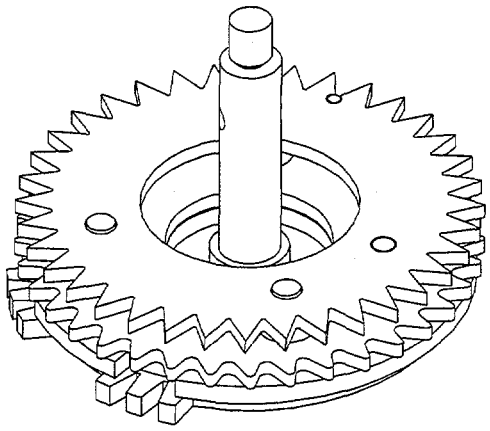


Figure 5: The pattern wheel assembly

Times were computed by running the planner, under a graphical user interface, on a Silicon Graphics Indigo2 200 MHz R4400 computer with Extreme graphics; times marked with an asterisk were run on an SGI Indigo2 100 MHz R4000 computer with Extreme graphics, without the graphical user interface. To calculate ACIS data size, the data for each distinct part is counted only once, regardless of the number of times that part appears in the assembly. Planning times are elapsed times necessary to load the pre-facetted data, identify all contacts in the assembly, and compute the plan.

The first figure shows a pattern wheel assembly, a main example handled by the original Archimedes system [30]. The pattern wheel has 13 parts, assembled unidirectionally, and is fastened together by laser welds. The pattern wheel was modeled with detailed welding specifications in the design module, and translated to ACIS format. The optimizing planner then determined an assembly plan for the pattern wheel. Since assembly reorientations and robot gripper changes are by far the slowest operations in the workcell, the optimality criterion given to the planner minimizes the number of these operations in the plan (as was discussed in section 2.4.3). The resulting plan was illustrated in a simulated workcell, and the plan was automatically translated to V+ code, using the library described above. The resulting robot program was executed in the workcell to assemble the pattern wheel. Laser welds were not performed automatically; instead, pre-welded subassemblies were substituted in the workcell when the program specified a laser weld.

To date, the pattern wheel is the only assembly which has exercised all modules of Archimedes 2.

Interested readers can retrieve color pictures of the examples, animations of sequences, illustrator output, and video of the robotic workcell assembling the pattern wheel on the World-Wide Web (`http://www.sandia.gov/2121/archimedes/archimedes.html`).

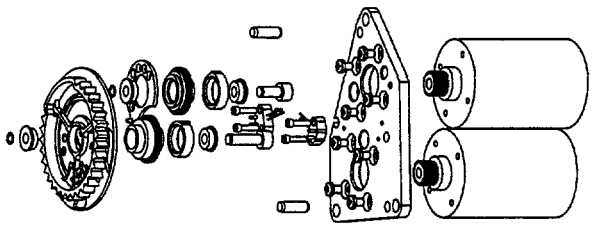


Figure 6: The discriminator

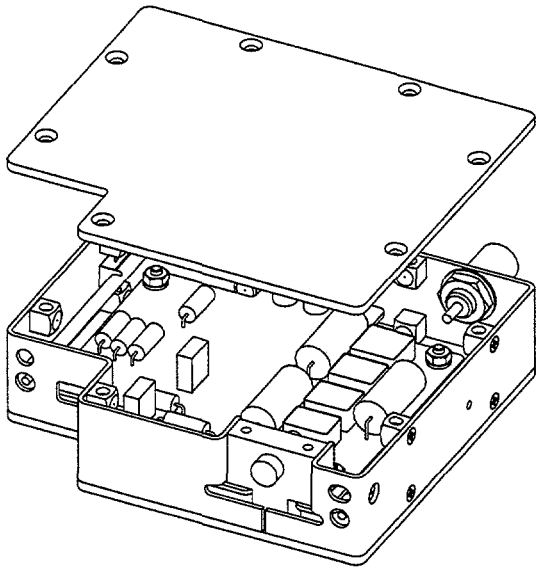


Figure 7: The Rockwell assembly

4 Lessons Learned during System Construction

This section describes what we learned in the course of constructing the Archimedes 2 system, and how these lessons affected subsequent development. Only the effect on the existing system is discussed; future work prompted by these experiences is the topic of the next section.

Algorithmic methods more successful than expected We had originally planned to use the ACIS volume-sweeping and volume intersection functions to determine if non-infinitesimal motions of parts will cause collisions. The computational cost of these checks was recognized, likely making exhaustive search of the space of assembly sequences im-

	N	MB	GE	OP
switchtube	5	0.2	7 sec	8 sec
pattern wheel	13	0.5	14	36
sprytron	14	1.1	17	33
neutron generator	35	1.5	51	—
Rockwell	78	3.2	167	—
fuel pump	36	3.6	175	—
discriminator	42	4.1	77	96*
locker	109	4.5	45*	—
seeker	70	6.1	120*	—
accelerometer	79	16	167	—
door lock	27	41	16 min*	—

Table 2: Planning times for various assemblies. "N" is the number of parts in the assembly, "MB" is the megabytes of ACIS data, "GE" is the geometric engine planning time, and "OP" is the optimizing planner planning time. An "*" indicates timing under different conditions; see text.

practical. We planned to use heuristic methods to prune away large areas of the search space.

However, the hardware-based collision-checking is fast enough that, for small assemblies, exhaustive search is effective. Therefore, we did not pursue the pruning heuristics.

Feature recognition replaced by user-supplied declarations This is an issue of finding the right balance between human and machine effort, as was discussed in section 2.2.

Difficulty of workcell programming To no one's surprise, programming the robot was a difficult and arduous task. In order to complete the path from CAD model to assembly sequence to robot program, we had to develop a library of robot functions that are the target of the translation process. Translating to the lowest-level robot commands is simply not practical at the present stage of technology; it would require solving several difficult research problems, including fine-motion planning and grasp planning.

The library includes routines for locating parts with the vision system, changing tools, acquiring parts, and placing parts. The last two classes of routines are parameterized according to the locations of the parts and their target locations. Once again, this is a question of finding the right tradeoff between effort on the part of the computer and effort on the part of a human. In this case, the machine effort is in the production of a robot program, and the human effort is in the writing of the robot function library.

5 Future Technical Work

5.1 More Constraints, Greater Realism

The assembly planning problem can be addressed at many levels of detail. Fully specified assembly plans must satisfy many constraints and specify many details, ranging from part accessibility, to stability and fixturing, to fine motion plans, to the people or technology targeted to assemble the product. A more aggressive definition would also include factory layout and scheduling, inventory control, and many other factors. In industry, a detail in any of these areas might drive a choice of assembly plan [27]. Hence to be useful, an assembly planning system must somehow enable generation of the "right" plan according to all these constraints.

However, most of these are active areas of research in themselves. Even if techniques existed to evaluate them, constructing a system that integrated all the constraints on assembly plans would be a huge effort, and it is not at all clear how they should be structured. Hence researchers in assembly planning are placed in the uncomfortable position of trying to integrate many not-fully-understood constraints, and yet produce useful output.

There are three major problems associated with this strategy of making assembly planners more realistic by incorporating more constraints: identification of appropriate constraints; finding a good computer representation of them; and structuring the planner to find plans satisfying all of them. We will discuss these problems in general terms, and then describe the constraints we have chosen to address in our near-term plans.

Identification of Constraints Several types of data are not typically captured in CAD models of assemblies, yet are critical to assembly planners. Often the information is called out on the final drawings (like weld sites), or implicit in part IDs (screw threads), but not represented in computer-accessible form. The design module captures some of this information currently, and we are adding more richness to our models. While it is straightforward to add this capability to a modeler, data coming from industrial users typically does not include the information needed for planning. Creating auxiliary data for industrial-size assemblies can be quite time-consuming, and it would be preferable to have the information in the model or derive it automatically (as the geometry engine does for simple contacts).

Encoding of Constraints Much assembly planning research has focused on languages to describe general assembly constraints (see e.g. [2, 14, 36]). A common type of assembly constraint is a *precedence constraint*, which is a logical relation specifying partial orders of assembly operations or states. However, it seems that many manufacturing constraints will resist being efficiently expressed as precedence constraints. This leaves the twin problems of rigorously encoding complex constraints and structuring the system in an efficient manner unresolved. Future work in the field must either address these problems head-on, or work in their shadow.

Structuring the Planner Several types of plan details are entered by hand in the current system, including gripper design and grasp planning, fixture design, and motion strategies for part mating. Gripper design, grasp planning, and fixture design are closely related areas, and have been widely studied for holding single parts (see e.g. [5, 7, 16, 26, 29, 31]). However, stability of and holding assemblies has received much less attention [25, 37], and these methods are for the most part not ready to be applied automatically. Similarly, research in automated fine-motion planning [8, 18, 23] has yet to find practical application. But the most important, and unaddressed, question is how to successfully integrate these techniques into the assembly planning process, where constraints flow both ways between the sequence and the tooling and motions used to accomplish it.

The search strategy in the current system proceeds top-down only: the system has no recourse when an assembly sequence generated by the optimizing planner cannot be translated to V+ due to additional constraints (such as the need to grasp subassemblies) that only appear in the translator. The obvious solution is simply to backtrack and generate another sequence, but this will quickly become computationally impractical. Perhaps better is to integrate these constraints into the initial sequence generation; however, this does not answer the question of how to structure a large system that must include so many types of constraints.

The A* search algorithm currently employed in the optimizing planner quickly becomes impractical when more than a small number of parts are present. We have not decided how to address this problem, but it is clear that optimality will have to be sacrificed, as for instance in [6].

5.1.1 Tool Accessibility

We have already begun work on using the constraint that all assembly operations requiring tools permit access of the tool to the point of application. Currently, the tools required for an operation is given in the auxiliary file for a subassembly, and all operations are checked for tool accessibility before allowing them to appear in a plan.

5.1.2 Workcell Representation

Work is just now beginning on representing the capabilities of workcells. Operations that an assembly apparatus can perform directly affect the quality of any assembly plan. For example, if a part is to be inserted horizontally, but the apparatus only allows vertical insertions, the target subassembly must be rotated and fixtured in its new pose. Such operations can be grouped together to reduce the total cost of the plan, of be eliminated entirely. But little is known about how to represent a workcell's capabilities in a way that can be used by the planner.

5.1.3 Grasp Planning

The need to grasp and manipulate parts and subassemblies is an important constraint in choosing an assembly sequence. The optimizing planner currently associates a gripper

with each part and minimizes the number of gripper changes in the chosen sequence, but otherwise the current system does not reason about grasping at all. Specifically missing is any capability to determine grasping positions, or whether a part can even be grasped in its current position. As a result, the generated assembly plans can often be unrealistic or even impossible.

Connected with our research on tools (above), we are adding geometric grasp reasoning capabilities to Archimedes. The first version will simply find grasp positions for a set of three standard robotic grippers: a parallel-jaw gripper, a suction gripper, and a vertical-chuck gripper. Once the initial grasp-planning library is working, we will consider more advanced topics. These might include determining a minimum set of grippers to assemble an object, minimizing gripper changes in assembly plans, or determining an assembly sequence for a multi-cell robotic assembly line such that each cell need have as few grippers and gripper changes as possible.

5.2 Assembly Modeling

In typical CAD systems today, assemblies are modeled as individual parts, placed in their relative locations. This view of assemblies overconstrains certain parts that are free to move when assembled; however, very little is known about non-monotone planning to take advantage of these freedoms [11]. It also causes inconsistencies for parts that deform when assembled—interference-fit parts intersect, and springs intersect with other parts—and often for threaded contacts as commonly modeled (cylinders at the outside of the male threads and the inside of the female threads). These inconsistencies in the assembly models cause severe problems for collision-detection algorithms.

This problem of inconsistent data for analysis is not particular to assembly planning; many other types of computer analysis, such as finite element methods and automatic NC program generation, have suffered from incomplete CAD data. In general, CAD systems create data for certain purposes—generating drawings was the first and still the most important purpose, but automated analysis is coming into wider use. When automated assembly planning presents sufficient value, engineers will enter models accurate enough to enable it. Work in PDES/STEP [15, 21] promises to make representations of such information standard, although in practical terms it will be many years before the critical assembly modeling capabilities of STEP are widely available.

5.3 Interactivity

Archimedes 2 includes little facility for user interaction. All user input to model the assembly and describe constraints is performed up front, in the design module; this input only describes features of the product itself. The system should allow designer-intended assembly plans or plan fragments to be entered, and allow interactive choices of assembly plans later in planning due to manufacturing constraints unknown to the programs. Methods to address these issues exist, but are not perfect. They either require editing a huge set of sequences [2], require constraints to be specified in advance [3, 10], or only apply to certain

geometric details [34]. Techniques that allow an engineer to richly and fully participate during the planning process are needed.

6 Technology Deployment

The Archimedes assembly planning system has met with very high levels of interest among possible customers within Sandia, at other government agencies, and in industry and universities. At the present time, however, it has too many bugs, it is not robust enough, and its user interface is not sufficient to support such use. We are currently addressing these limitations in hopes of deploying the software to engineers.

Under the auspices of the Technology Information Environment for Industry (TIE-In) program, a graphic user interface (GUI) has been developed for the planner. The GUI is programmed using Tcl/Tk and Xwindows, with animation routines written in OpenGL. It allows a user to load an assembly from a file browser, get graphic analysis results during planning, and see animations of the resulting sequences. Status and progress information is displayed in a progress window. The planner is slated to be available to remote users over TIE-In when the GL graphics can function over the network, and several corporations have indicated interest in testing the system at that time for possible use.

A more primitive version of the planner has been delivered to the robotics research group of Prof. George Bekey at the University of Southern California; the planner has determined assembly sequences for several robotically assembled fixture setups as part of a larger project. In addition, we are currently negotiating with a research group at Carnegie Mellon University who wish to incorporate the software into their system.

If interest warrants, eventually the technology will need to become a product. We have approached CAD tool vendors and robotic simulator suppliers to gauge interest in marketing an assembly planning system, and have received limited but encouraging interest. Once Archimedes has been tested and found useful by end users, and improved as a result of the feedback we receive from those users, this interest is bound to increase.

7 Conclusion

We have described Archimedes 2 and its capabilities, as determined by experimentation on models of real mechanical assemblies. Based on its performance, we believe that it represents the state of the art in implemented assembly planning systems, at least in terms of the complexity of assemblies for which plans have been successfully computed. Two aspects of the system, in particular, have allowed us to achieve these results: the fast collision detection algorithm; and the declaration of part matings that are threaded or require part deformation, in place of sophisticated feature-recognition software. This latter point suggests that human input to the assembly planning process is critical, and that future research should address finding the proper balance between human and machine effort.

A Appendix: Project Metrics

Publications and presentations resulting from the project: [19], [1], [9].

Patent and Invention Disclosures resulting from the project: Technical Advance disclosure SD-5421, "A Polyhedral collision-detection technique using a Z-buffer", 2/4/94.

Software copyrights resulting from the project: DOE Memo dated August 12, 1994, granting Sandia permission to assert its copyright in software entitled "Sandia Assembly Planner".

Employess recruited to work on the project: 4 MTS (2 external), 1 TA (lateral)

Student involvement in the project: A graduate student in Electrical Engineering at Stanford worked on the project during the summer of 1995.

Follow-on work: See section 6.

References

[1] A. L. Ames, T. L. Calton, R. E. Jones, S. G. Kaufman, C. A. Laguna, and R. H. Wilson. Lessons learned from a second generation assembly planning system. In *Proc. of the IEEE Intl. Symp. on Assembly and Task Planning*, 1995.

[2] D. F. Baldwin, T. E. Abell, M.-C. M. Lui, T. L. De Fazio, and D. E. Whitney. An integrated computer aid for generating and evaluating assembly sequences for mechanical products. *IEEE Trans. on Robotics and Automation*, 7(1):78–94, 1991.

[3] N. Boneschanscher and C. J. M. Heemskerk. Grouping parts to reduce the complexity of assembly sequence planning. In E. A. Puente and L. Nemes, editors, *Information Control Problems in Manufacturing Technology 1989: Selected Papers from the 6th IFAC/IFIP/IFORS/IMACS Symposium*, pages 233–238. Pergamon Press, 1989.

[4] A. Bourjault. *Contribution à une approche méthodologique de l'assemblage automatisé: élaboration automatique des séquences opératoires.* PhD thesis, Faculté des Sciences et des Techniques de l'Université de Franche-Comté, 1984.

[5] R. C. Brost and K. Y. Goldberg. A complete algorithm for synthesizing modular fixtures for polygonal parts. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 535–542, 1994.

[6] S. Chakrabarty and J. Wolter. A hierarchical approach to assembly planning. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 258–263, 1994.

[7] M. R. Cutkosky. *Robotic Grasping and Fine Manipulation.* Kluwer Academic Publishers, 1985.

[8] M. A. Erdmann. Using backprojections for fine motion planning with uncertainty. *Intl. J. of Robotics Research*, 5(1):19–45, 1986.

[9] L. J. Guibas, D. Halperin, H. Hirukawa, J.-C. Latombe, and R. H. Wilson. A simple and efficient procedure for polyhedral assembly partitioning under infinitesimal motions. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 2553–2560, 1995. Extended version submitted to Computational Geometry and Applications.

[10] J. M. Henrioud and A. Bourjault. LEGA: a computer-aided generator of assembly plans. In *[12]*, pages 191–215.

[11] R. L. Hoffman. Automated assembly in a CSG domain. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 210–215, 1989.

[12] L. S. Homem de Mello and S. Lee, editors. *Computer-Aided Mechanical Assembly Planning*. Kluwer Academic Publishers, 1991.

[13] L. S. Homem de Mello and A. C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Trans. on Robotics and Automation*, 7(2):228–240, 1991.

[14] Y. F. Huang and C. S. G. Lee. Precedence knowledge in feature mating operation assembly planning. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 216–221, 1989.

[15] ISO. *ISO 10303: Product Data Representation and Exchange*, 1994.

[16] J. Jones and T. Lozano-Pérez. Planning two-fingered grasps for pick-and-place operations on polyhedra. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 683–688, 1990.

[17] H. Ko and K. Lee. Automatic assembling procedure generation from mating conditions. *Computer Aided Design*, 19(1):3–10, 1987.

[18] J.-C. Latombe, A. Lazanas, and S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, 52:1–47, 1991.

[19] J.-C. Latombe and R. Wilson. Assembly sequencing with toleranced parts. In *Proc. of the Third Symp. on Solid Modeling and Applications*, 1995. Revised version submitted to Trans. on Robotics and Automation.

[20] L. I. Lieberman and M. A. Wesley. AUTOPASS: An automatic programming system for computer controlled mechanical assembly. *IBM J. of Research and Development*, 21(4):321–333, 1977.

[21] T.-H. Liu and G. W. Fischer. Developing feature-based manufacturing applications using PDES/STEP. *Concurrent Engineering: Research and Applications*, 1(1):39–50, 1993.

[22] T. Lozano-Pérez, J. L. Jones, E. Mazer, P. A. O'Donnell, W. E. L. Grimson, P. Tournassoud, and A. Lanusse. Handey: A task-level robot system. In *Robotics Research: The Fourth Intl. Symposium*, pages 29–36. MIT Press, 1988.

[23] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *Intl. Journal of Robotics Research*, 3(1):3–24, 1984.

[24] T. Lozano-Pérez and P. H. Winston. LAMA: A language for automatic mechanical assembly. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence*, 1977.

[25] R. Mattikalli, D. Baraff, and P. Khosla. Finding all gravitationally stable orientations of assemblies. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 251–257, 1994.

[26] B. Mishra, J. T. Schwartz, and M. Sharir. On the existence and synthesis of multifinger positive grips. *Algorithmica*, 2:541–558, 1987.

[27] J. L. Nevins, D. E. Whitney, T. L. De Fazio, A. C. Edsall, R. E. Gustavson, R. W. Metzinger, and W. A. Dvorak. *Concurrent Design of Products and Processes.* McGraw-Hill, 1989.

[28] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing, 1980.

[29] J. Pertin-Troccaz. Grasping: A state of the art. In Khatib, Craig, and Lozano-Pérez, editors, *The Robotics Review 1.* MIT Press, 1989.

[30] D. R. Strip and A. A. Maciejewski. Archimedes: An experiment in automating mechanical assembly. In *Proc. of the 11th ASME Intl. Conf. on Assembly Automation*, 1990.

[31] A. S. Wallack and J. F. Canny. Planning for modular and hybrid fixtures. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 520–527, 1994.

[32] R. H. Wilson. A framework for geometric reasoning about tools in assembly. Submitted to 1996 IEEE Int. Conf. on Robotics and Automation.

[33] R. H. Wilson. *On Geometric Assembly Planning.* PhD thesis, Stanford Univ., March 1992. Stanford Technical Report STAN-CS-92-1416.

[34] R. H. Wilson. Minimizing user queries in interactive assembly planning. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, volume 2, pages 322–327, 1993.

[35] R. H. Wilson and J.-C. Latombe. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71(2):371–396, 1994.

[36] J. D. Wolter, S. Chakrabarty, and J. Tsao. Mating constraint languages for assembly sequence planning. *IEEE Trans. on Robotics and Automation.* To appear.

[37] J. D. Wolter and J. C. Trinkle. Automatic selection of fixture points for frictionless assemblies. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation*, pages 528–534, 1994.

DISTRIBUTION:

1	MS-0188	LDRD Office, 4523
1	1006	S. G. Kaufman, 9671
1	1008	R. H. Wilson, 9621
1	1008	R. E. Jones, 9621
1	1008	T. L. Calton, 9621
1	1010	A. L. Ames, 9622
1	1002	P. J. Eicker, 9600
1	1003	R. D. Robinett, 9611
1	1008	D. R. Strip, 9621
1	1010	M. E. Olson, 9622
1	1004	R. D. Palmquist, 9651
1	1004	D. S. Horschel, 9661
1	1006	P. Garcia, 9671
1	1007	A. T. Jones, 9672
1	9018	Central Technical Files, 8523-2
5	0899	Technical Library, 4414
1	0619	Print Media, 12615
2	0100	Document Processing, 7613-2 For DOE/OSTI