Micro Planning for Mechanical Assembly Operations

S. K. Gupta

C. J. J. Paredis

P. F. Brown

Robotics Institute Carnegie Mellon University Pittsburgh, PA 15213 Institute for Complex Engineered Systems Carnegie Mellon University Pittsburgh, PA 15213

Abstract

Significant advances have been made in the area of macro planning for assembly operations (i.e., dividing a product into sub-assemblies, determining the sequence of assembly operations). On the contrary, the state of the art in micro planning (i.e., tool selection, path planning for tool and part movements) is rather primitive. To generate a realizable assembly plan, both macro planning and micro planning problems need to be solved. Tooling considerations are an important component of the micro planning problem. Therefore, we present a methodology for modeling assembly tools, selecting tools for assembly operations, and generating detailed tool movements. Techniques described in this paper can be combined with macro planning techniques to result in a complete assembly planner.

1 Introduction

Developing high-performance electro-mechanical products is a very challenging task. To improve the efficiency and to reduce the product weight and volume, designers have to pack a large number of components in a very small space. At the same time, there needs to be sufficient room for performing assembly and disassembly operations. These conflicting requirements make design of electro-mechanical products a highly iterative process. Quite often, the product development process includes physical prototyping in the design loop to verify proper functioning and ease of assembly. Physical prototyping tends to slow down the product development process and constrain the number of design alternatives that can be examined considerably.

We are developing an assembly planning and simulation environment for electro-mechanical products. Our goal in this project is to develop a high fidelity assembly simulation and visualization systems that can detect assembly related problems without going through physical mock-ups. To assist the user in creating these simulations, we have developed a micro planning system.

Depending on the level of plan detail, assembly planning systems can be divided into two different types: macro planners and micro planners. Macro planners deal with the high level planning decisions, such as dividing a product into sub-assemblies, or determining the sequence of assembly operations. Micro planners, on the other hand, deal with the low level planning decisions, such as selecting appropriate tools, or planning detailed tool and part movements. To create a completely automated assembly planning system both capabilities are necessary. A lot of work has been done in developing macro planners for assembly planning [6, 7, 3, 2, 12, 17, 24, 15, 20], but the area of micro planning is relatively under-explored. Therefore, we have chosen to focus on micro planning. We hope that our work can later be integrated with macro planners to create a complete assembly planning system. In the mean time, we intend to rely on designers/process engineers for macro planning while providing them with micro planning tools to quickly verify the high level plans.

This paper primarily describes our research in the areas of tool modeling, tool selection, and path planning for tool movements.

2 System Overview

As shown in Figure 1, our assembly toolkit consists of four major components: interface agents, synthesis agents, analysis agents, and a data manager. We have chosen to adopt a modular agent-based architecture for the following reasons:

• expandability: new synthesis and analysis agents can be easily added.

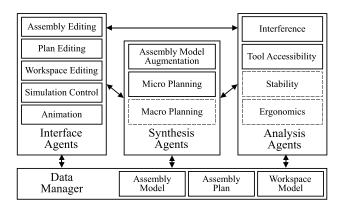


Figure 1: System architecture of the assembly toolkit. Dashed components have not yet been implemented.

- composability: the user can compose a group of analysis agents that is appropriate for the current design phase.
- performance: some synthesis and analysis algorithms are computationally expensive; one may want to distribute these agents over several networked workstations for parallel processing.

The following scenario illustrates the overall capabilities of our system and describes the function of each of the components. A designer creates an assembly design using a commercial CAD package. After saving the design in ACIS format, he imports the assembly into our system using the assembly editor. Next, the assembly model augmentation agent automatically builds a contact graph indicating the types of contacts (fixed, prismatic joint, revolute joint, etc.) between the parts of the assembly. Using feature recognition, it also recognizes assembly features on individual parts, e.g. a hexagonal slot on a screw; this information will later be used by the micro planner to select tools and determine tool paths. In collaboration with the process engineer, the designer then uses the workspace editor to create the work-cell in which the assembly operations will take place. They decide the layout of the environment and the positions of each of the parts and tools. Subsequently, the process engineer enters a high level plan (macro plan) using the plan editor. The micro planner automatically converts the high-level assembly operations and tooling information into low-level part and tool motions (micro plans). The animation viewer allows the process engineer to verify the assembly plan visually. For a more scrupulous analysis, he selects several analysis agents (e.g. interference or stability analysis) with the simulation controller, and composes them into a customized simulation. After several design and planning iterations, the process engineer generates a final animation of the assembly process and downloads it to the operator's desktop computer. The assembly operator can start assembling the parts immediately without the need for extensive training. He can randomly access any particular operation in the assembly sequence and interactively change the 3D viewpoint within the animation.

3 Tooling

The ability to account for tooling requirements is critical in both design and assembly of complex electro-mechanical products. The designers need to make sure there is no interference between tools and parts. Likewise, process engineers need to ensure that an assembly operator has enough room to manipulate the tools. Moreover, both the execution time and the quality of the assembly depend on the tool type and its particular application.

Without adequate software-aids, designers currently rely on physical prototypes to investigate tool accessibility issues. We are building micro planning software that helps designers and process engineers to select and evaluate tool applications within assembly plans.

3.1 Previous Work

To analyze tool accessibility in machining operations, two different approaches have been developed. The first approach checks for interference between the workpiece and the tool accessibility volume, which is defined as the non-cutting portion of the tool [10, 19, 8]. The second approach uses visibility maps [23, 14]; a machining operation for a particular face is selected such that the tool approach direction is inside the visibility region for that face. Both approaches have also been applied for coordinate measuring machines, to determine the accessibility of inspection points.

The tool accessibility problem has also been investigated for the assembly domain. Homem deMello and Sanderson's [4] model of assemblies include attachments, which describe fastening methods. Four types of attachment are considered: clip, pressure, screw, and glue. Even though the attachments can be used to generate disassembly sequences, the detailed tool movements required to carry out these attachment are not modeled. Diaz et al. [5] present a method that automatically determines the complexity of an assembly

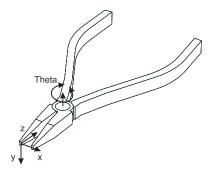


Figure 2: Tool representation for a plier including its application frame and degree-of-freedom.

task based on the tool use. The most detailed work is reported by Wilson [22]. Wilson developed a tool representation that includes the tool's use volume, i.e. the minimum space that must be free in an assembly to apply the tool. He further divides tools into pre-tool, in-tool, and post-tool classes, based on the time at which the tools are applied relative to when the parts are mated in the assembly operation. In addition, he provides algorithms for finding feasible tool placements.

3.2 Tool Representation

To enable reasoning about tool usage, we have developed tool models that capture the following three types of information.

Tool Geometry: To investigate accessibility issues, we need to represent the geometry of a tool. In general, we model tools as articulated devices. During the assembly process, one can control the joint values for each of the DOFs of the tool. As illustrated in Figure 2 and 3, we also define a tool application frame that indicates the tool's position relative to the part to which it is applied. Some tools can be used in more than one way. Therefore, we allow the same physical tool to correspond to multiple logical tools. For example, the Allen wrench in Figure 4 can be treated as two different logical tools each with its own application frame and motion definition.

Tool Parameters: For a given assembly operation, the size and shape of the tool need to match the size and shape of the part feature to which it is applied. To check whether this condition is satisfied without having to rely on geometric reasoning with solid models, the size of the tool is abstracted in the tool parameters. As illustrated in Figure 5, a flathead screwdriver is characterized by the length, width, and height of its tip. These parameters are compared to the parameters



Figure 3: Tool representation for a screw driver.

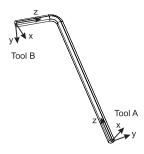


Figure 4: Two logical tools corresponding to the same physical tool.

of the corresponding part feature in the applicability condition.

Tool Use: The final component of our tool model lists, for each type of assembly operation supported by the tool, the sequence of motions describing the proper tool use. Some of the types of assembly operations that are currently supported are: position, screw, unscrew, tighten, loosen, reorient, fit, and extract. A tool will typically support one or more of these operations. We have also modeled a logical tool called "no_tool" that has no physical tool attached to it and supports all of the above operations.

As shown in Figure 5, the motion primitives are expressed in TCL syntax and may contain references to both the part and tool parameters. This parametric description allows complicated tool movements to be expressed as a simple sequence of elementary motions. Some elementary motions, such as relative and absolute linear moves of parts or tools, translate directly into low-level simulation primitives; others, such as "screw with retraction," require a large number of simulation primitives. The "screw with retraction" motion-macro divides a single screwing operation into multiple applications over 60 or 120 degrees each. This

```
{ Tool
  { Name screwdriver }
  { Body
    { URL screwdriver1.asm }
    { Transform identity }
  { PartFeatures
    { Name FlatSlot }
    { Name Thread }
  { Parameters
    { Length 3 }
    { Width 0.8 }
    { Height 1.2 }
  }
  { ApplicabilityCondition
    # TCL script
    "expr $Width < $FlatSlot(Width) &&
     $Height > $FlatSlot(Depth)"
  { Operation { Type screw }
    # TCL script
    "# engage motion
     AbsMoveTool $FlatSlot(Reference)
                 RelativeTo $part(transform);
     RelMoveTool Transform translation
                 0 0 - $FlatSlot(Depth);
     Attach $tool(name) $part(name);
     # operate motion
     RelMoveTool Transform screw 0 0 0 0 0 1
         [expr -360*$Thread(Depth)*$Thread(hand)/
               $Thread(Pitch)] -$Thread(Depth);
     # disengage motion
     Detach $tool(name) $part(name);
            RelMoveTool Transform translation
            0 0 $FlatSlot(Depth);"
  }
  { Operation { Type unscrew }
    # TCL script
    "# engage motion
     AbsMoveTool $FlatSlot(Reference)
                 RelativeTo $part(transform);
     RelMoveTool Transform translation
                 0 0 - $FlatSlot(Depth);
     Attach $tool(name) $part(name);
     # operate motion
     RelMoveTool Transform screw 0 0 0 0 1
         [expr 360*$Thread(Depth)*$Thread(hand)/
               $Thread(Pitch)] $Thread(Depth);
     # disengage motion
     Detach $tool(name) $part(name);
            RelMoveTool Transform translation
            0 0 $FlatSlot(Depth);"
}
```

Figure 5: A file describing a flathead screwdriver.

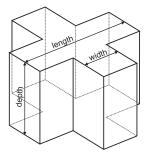


Figure 6: A Phillips slot feature.

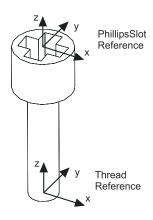


Figure 7: Phillips slot feature on a screw.

is useful in case a wrench is used in a confined area in which it cannot complete a full 360-degree rotation.

3.3 Tool selection

Whether a tool is applicable depends on the type of operation for which it is to be used, and on the features of the part to which it is applied. For example, a Phillips screwdriver can only be used for screwing in a screw with a Phillips slot of the appropriate dimensions. For a given assembly operation, a feasible tool is selected according to the following steps:

- 1. Consider only those tools for which the given assembly operation is supported.
- 2. Verify that the part to which the tool is applied has all the required features listed in the tool model (the definitions of the part features are described in the next section)
- 3. Evaluate the tool applicability condition to verify that the dimensions of the tool match the dimensions of the corresponding part feature.
- 4. Finally, for parts containing symmetry, determine the initial position of the tool and part such that

no collisions occur during the execution of the motion script; this is a planning problem that will be addressed in Section 4.

3.4 Part feature recognition

We have developed an algorithm that automatically recognizes the following assembly features: FlatSlot, PhillipsSlot, HexagonalSlot, HexagonalBoss, and CircularHole.

The feature recognition algorithm is quite generic because of our canonical representation of features. The representation consists of a parametric base face (with a particular surface type, number of edges, interior angles, and accessibility condition) and a set of neighboring faces. The algorithm loops over all the faces in the solid model and compares each of them with the parametric representation of the features' base face. If there is a match and the base face needs to be accessible, the algorithm checks for intersections between the normally swept volume of the base face and the rest of the part. Finally, the neighboring faces of the base face are matched with the neighboring faces of the feature template. If all of the above conditions are met, the algorithm extracts features' reference frame and the values of its parameters.

Two additional features are required to support automatic tool generation for all the assembly operations: Thread and AdhesiveFace. These features do not have any distinguishing geometric characteristics (a thread is commonly modeled as a plain cylinder); therefore, the user has to identify them and provide their relevant parameters. As an example consider the PhillipsSlot feature defined in Figure 6 and shown in Figure 7 as part of a screw. The parameter values and reference frame are listed in the description file in Figure 8.

4 Planning

As described in the previous section, for a given tool and assembly operation, the tool motion script models the motion sequence of the tool required to execute the assembly operation. However, to specify the tool motion completely, there are still two main issues that need to be addressed:

Determination of the tool application position: Many part and tool features contain an element of symmetry. For instance, the thread of a screw can be considered axially symmetric, while the tool feature of an Allen wrench has hexagonal symmetry. As a result,

```
{ Part
  { Color steelblue }
  { Transform identity }
  { Material
    { Name steel }
    { Density 7.8 }
  { Feature
    { Type PhillipsSlot }
    { Width 1.5 }
    { Depth 1.2 }
    { Length 6.5 }
    { Reference Transform homogeneous
                0 0 4 0 -1 0 1 0 0 }
  { Feature
    { Type Thread }
    { Pitch 1.5 }
                      # metric thread
    { Depth 20 }
    { Hand 1 }
                      # right-handed
}
```

Figure 8: A file describing a part with its features.

a tool may be applied in a (possibly infinite) number of different application positions; sometimes, the choice of application position determines whether an assembly operation fails or succeeds. For instance, certain application position may cause a collision during the execution of the tool's motion script. It is the task of the micro planner to determine a feasible application position.

Determination of the approach/return path: In most (dis-)assembly sequence planners, it is assumed that parts and tools can be extracted along a straight line from their assembled position to infinity [21, 12]. In real world assemblies, part and tool trajectories commonly consist of sequences of translations and rotations along different directions. It is the task of the micro planner to determine this motion sequence.

4.1 Tool application position

For symmetric tools and/or parts, there may exist multiple tool application positions. Wilson [22] recognized this and defined the number of DOFs for a tool as the number of independent directions along which a tool can be repositioned while performing a given assembly operation. For instance, when applying glue with a glue gun, the orientation of the gun can be chosen freely, resulting in 3 DOFs. It is important to notice that the number of DOFs depends

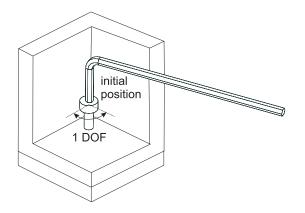


Figure 9: Finding the initial position for a wrench.

not only on the tool but also on the part to which the tool is applied; an open-end wrench does not have any symmetry, but when applied to a bolt with axial symmetry, the operation still has one DOF. Figure 9 shows an example.

The goal is to find values for each of the DOFs so that the resulting application position does not cause any collisions during the execution of the tool's motion script. Strictly speaking, the goal should be to find not only the initial application position but the complete tool path, including intermediate retractions and articulations of the tool. This is a very complicated problem that we have yet to address. In this paper, we will limit ourselves to finding the application position for 1-DOF assembly operations.

Finding a tool application position with 1-DOF requires a search through a 1-dimensional search space. To guide the search, we have chosen the following heuristic: the fraction of time during which the tool collides with any other part in the assembly or environment. This heuristic is computed by performing a simulation of the motion script starting from the current application position, and checking for interference at discrete time intervals. Note that the heuristic may introduce multiple local minima, so that a global search method is required. We have chosen the P^* algorithm introduced by Zilinskas [25]. P^* combines a one-stage Bayesian algorithm with Brent's local minimization, resulting in smart global coverage and accurate local refinement. Moreover, the method does not require gradient information and is relatively easy to implement.

4.2 Tool approach path

To move a tool from its initial position in the workspace environment to the application position, re-

quires in general a sequence of translations/rotations.

To generate these complicated paths, one could rely on the user's geometric intuition. Indeed, humans tend to be rather good at finding a collision free path between an initial and final position. However, there is currently no convenient mechanism available to enter a 6-DOF path into the computer; specifying via-points textually is tedious and error-prone, graphical entry is difficult because a mouse has only two DOFs, and VR environments with 6-DOF hand tracking are still very expensive and therefore not readily accessible.

It is our goal to relieve the user from the path planning task all together, and generate tool and part paths automatically. The 6-DOF path planning problem is very challenging, however, due to the following characteristics. First, in the application position, the tool tends to be in contact with the part to which it is applied. That means that in the configuration space the goal configuration may be almost completely surrounded by obstacles, which is a situation that is typically difficult to handle for path planning algorithms. Second, one only computes a path once for each part in the assembly. This means, that one cannot amortize any of the pre-computations required by some of the path planning algorithms (e.g. pre-computation of the configuration space).

Over the years, many different path planning algorithms have been developed [13]: roadmap methods, cell decomposition methods, potential field methods, etc. Most algorithms require the computation of the configuration space representation of obstacles which is very expensive for a 6-DOF C-space cluttered with obstacles, as in assembly planning. Moreover, this computation cannot be amortized, because the C-space changes dramatically when tools and parts are moved.

Several path planning algorithms have been developed specifically for assembly planning [1, 18]. Our implementation is based on a group of randomized path planners [1, 11, 16]. These algorithms do not require the computation of the C-space; they consist of the following components: a simple potential field method, generation of random via-points, creation of a roadmap graph, and search of the roadmap. For a detailed description of these algorithms refer to [11, 16].

We have implemented a simple potential field method that is an extension of the method described in [16]. The method lets the tool make small moves to 36 neighboring positions (6 pure translations, 6 pure rotations, and 24 combined rotations/translations), and ranks these positions according to their distance from the goal position. The tool is moved to that

direct neighbor that is closest to the goal without colliding with any obstacles. The algorithm terminates when the goal is reached or when no neighboring positions closer to the goal are collision free.

To check for collision, we use a combination of RAPID and ACIS. RAPID is a fast collision detection algorithm based on oriented bounding-box trees [9]. Because it uses a faceted representation of the tools and objects, it often detects collisions between objects that are in contact but do not really intersect. For instance, when rotating a cylindrical object inside a cylindrical hole of the same diameter, RAPID will always detect a collision except when the facets of the hole happen to line up perfectly with the facets of the cylinder. Since contact situations are common in assembly, we use the ACIS geometric kernel to check for intersections between the exact solid models whenever RAPID detects a collision between the faceted models. In this way, we are able to check for collisions rapidly and accurately.

The creation of the roadmap based on randomly generated via-points is unacceptably slow for our purposes. Instead, we have defined a set of via points that yields feasible paths in most cases. These points are located at the corners of an expanded bounding box around the subassembly of the parts that are being mated. If this selection does not result in a connected roadmap, then the user is asked to identify promising via-points using a graphical user interface, until a feasible path is found.

Most of the time, this simple algorithm is able to find feasible paths relatively quickly for a single tool movement. By performing the computation in the background, while the user is specifying the next assembly operation, the path planning does not cause significant delays. Only when the user changes or inserts an assembly operation towards the beginning of the assembly plan becomes path planning a bottleneck. Indeed, a change in an assembly operation may change the position of one or more objects in the workspace which invalidates all subsequent tool paths.

5 Conclusions

In this paper, we have present a methodology for modeling tools, selecting tools, and generating detailed tool paths for mechanical assembly operations. Our approach improves the state of the art in micro planning for mechanical assemblies in the following manner:

- Tools are modeled as articulated devices. This allows us to consider tools that use articulation to perform assembly operations. Our notion of articulated devices can be easily extended to include a model of the assembly agent (human hand, or robot) that is manipulating the tool.
- We model the motion of tool during its use. This allows us to perform very accurate accessibility checks and select the appropriate tool for a given assembly operation.
- We have developed tool selection and path planning techniques to automatically select tools and generate detailed tool movements for a wide variety of assembly operations.

We believe that the research presented in this paper is a step towards an automated micro planner and can eventually be combined with macro planning to create a complete assembly planner.

Acknowledgments

This research was funded in part by DARPA under contract ONR #N00014-96-1-0854, by Raytheon Company, the Robotics Institute, and the Institute for Complex Engineered Systems at Carnegie Mellon University.

References

- [1] Nancy M. Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, Minneapolis, MN, April 1996. IEEE.
- [2] L Homem de Mello. Task Sequence Planning for Robotic Assembly. PhD thesis, Carnegie Mellon University, May 1989.
- [3] L Homem de Mello and A Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. In *IEEE International Conference on Robotics and Automation*, pages 56-61, May 1989.
- [4] L.S. Homem de Mello and A.C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, 7(2):228-240, April 1991.

- [5] Antonio Diaz-Calderon, D. Navin-Chandra, and Pradeep K. Khosla. Measuring the difficulty of assembly tasks from tool access information. In Proceedings of the IEEE International Symposum on Assembly and Task Planning, pages 87-93, 1995.
- [6] T De Fazio and D Whitney. Simplified generation of all mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, 3(6):640-658, Dec 1987.
- [7] T De Fazio, D Whitney, Man-Cheung Max Lui, T Abell, and D Baldwin. Aids for the design of choice of assembly sequences. In *IEEE Interna*tional Conference on Systems Machines and Cybernatics, Nov 1989.
- [8] Daniel Gaines and Caroline Hayes. A constraint-based algorithm for reasoning about the shape producing capabilities of cutting tools in machined part. In ASME Design for Manufacturing Conference, Sacremento, CA, September 1997.
- [9] S. Gottschalk, M. C. Lin, and D. Manocha. OBBtree: a hierarchical structure for rapid interference detection. In *Proceedings of ACM Sig*graph '96, 1996.
- [10] S.K. Gupta. Automated Manufacturability Analysis of Machined Parts. PhD thesis, University of Maryland, College Park, MD, September 1994.
- [11] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566-580, August 1996.
- [12] P Khosla and R Mattikalli. Determining the assembly sequence from a 3-d model. *Journal of Mechanical Working Technology*, pages 153–162, Sep 1989.
- [13] Jean-Claude Latombe. Robot motion planning. Kluwer Academic Publishers, Norwell, MA, 1991.
- [14] Putta Laxmiprasad and Sanjay Sarma. A feature free approach to 5-axis tool path generation. In ASME Design for Manufacturing Conference, Sacremento, CA, September 1997.
- [15] Joseph Millner, Stephen Graves, and Daniel Whitney. Using simulated annealing to select

- least-cost assembly sequences. In *IEEE International Conference on Robotics and Automation*, pages 2058–2063, San Diego, May 1994.
- [16] Mark. H. Overmars and Petr Svestka. A probabilistic learning approach to motion planning. In Proceedings of the First Workshop on the Alogrithmic Foundations of Robotics, pages 19–37. A. K. Peeters, Boston, MA, 1994.
- [17] A Sanderson, L Homem de Mello, and H Zhang. Assembly sequence planning. AI Magazine, 11(1):62-81, Spring 1990.
- [18] Achim Schweikard and Fabian Schwarzer. General translational assembly planning. In *Proceedings fo the 1997 IEEE International Conference on Robotics and Automation*, pages 612–619, Albuquerque, NM, April 1997.
- [19] Roger Stage, Mark Henderson, and Chell Roberts. A framework for representing and computing tool accessibility. In ASME Design for Manufacturing Conference, Sacremento, CA, September 1997.
- [20] A Swaminathan and K S Barber. Ape: An experience-based assembly sequence planner for mechanical assemblies. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 1278–1283, May 1995.
- [21] R. Wilson. On Geometric Assembly Planning. PhD thesis, Dept of Computer Science, Stanford University, March 1992.
- [22] Randall H. Wilson. A framework for geometric reasoning about tools in assembly. In Proceedings of the 1996 International Conference on Robotics and Automation, pages 1837–1844, Minneapolis, MN, 1996.
- [23] T. Woo. Visiblity maps and spherical algorithms. Computer-Aided Design, 26(1), 1994.
- [24] T. C. Woo and D. Dutta. Automatic disassembly and total ordering in three dimensions. ASME Transactions, Journal for Engineering for Industry, 113(2):207-213, 1991.
- [25] A. Zilinskas. Optimization of one-dimensional multimodal functions, algorithm AS133. Applied Statistics, 23:367–375, 1978.