Subassembly Identification and Motion Generation for Assembly : A Geometric Approach

Raju S. Mattikalli   Pradeep K. Khosla   Yangsheng Xu

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

1. Abstract

The long term goal of our research is to address the question: Can the given MSA be assembled automatically with the given facilities? In order to accomplish our goal we are developing techniques to model the Mechanical System/Assembly (MSA), the available facilities and the assembly process. In this paper, we address the first part of our goals and develop a methodology to automatically determine the assembly sequence from a 3-D geometric modeler description of the assembly. Our approach consists of automatically determining a set of assembly operations, through a disassembly procedure, that lead to the given assembly (MSA). We present our work on identifying subassemblies that can be assembled using combinations of translational and rotational motions.

2. Introduction

Software tools that evaluate the manufacturability of a design would allow the designer to anticipate potential problems that may arise during the manufacture of a product, i.e., they would help the designer to *Design for Manufacturability*. By providing useful feedback to the designer during the design of a product, these tools will help reduce the design-manufacture cycle time. In our work, we address the process of assembly. Feedback about assembly concerns could be as simple as a YES/NO answer regarding the assemblability of the design on the given facilities, or, it could be something more complex that would identify certain assembly tasks that cannot be carried out and include suggestions for redesign of the component. In either case, in order to be useful and practical, such tools should have complete models of the assembly and the manufacturing facility. A system that could critique a design and provide useful feedback for redesign is still far from the horizon, but substantial amount of present day research is being targeted towards the development of important parts of such a system.

We are developing a system that will accept a 3-D geometric description of an assembly and a model of assembly facilities, and reason about the assembly to answer the question: Can the given assembly be automatically assembled on the available facility? In order to answer this question we are addressing issues that will allow us:

- to create a suitable representation of the assembly,
- to generate the sequence of assembly operations,
- to create a representation of the facility,
- to reason about the assemblability, using the assembly operations and the facility model.

If the mechanical system can be assembled on the given facilities, each assembly task is automatically planned and programmed as a sequence of standardized actions corresponding to the the specific assembly task. These programs could be downloaded onto the real assembly facilities to perform the assembly. The block diagram in Figure 1 shows an overview of the research.

Our research group in the Engineering Design Research Center, at Carnegie Mellon, has developed a geometric modeler *Noodles* [3]. The work described in this paper utilizes *Noodles* to create 3-D representations of assemblies. Starting from this representation, we have developed a strategy, based on kinematic constraints, that automatically determines the assembly sequence of the given Mechanical System/Assembly (MSA). Subassemblies that satisfy stability criteria and which are accessible for manipulation are identified. By posing it as a problem of reasoning about changes in degrees-of-freedom of geometric subassemblies due to their movement within the assembly, sequences of rotational and translational motions can be determined. We have developed algorithms that compute such collision free motions. This work is an important step towards developing software tools to evaluate the assemblability of mechanical assemblies.

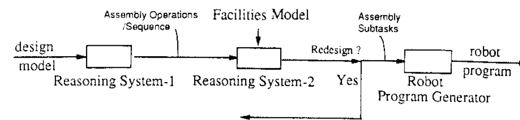

Figure 1: Research Overview

One of the early works in evaluating and reducing assembly costs is that by Boothroyd and Dewhurst [1]. They rate the efficiency of a design based on a classification of the various geometric features of the components comprising the assembly. These features are used to estimate the full cost of automation using hueristic knowledge. In our work, we propose to make assembly evaluations based on knowledge about the details of the assembly process. Using a *geometric model* of an MSA, we systematically determine the assembly tasks that assemble the MSA. Two principal steps are involved : (a) identifying subassemblies (a topologically connected group of components), and (b) finding detailed assembly motions of each subassembly. Knowledge about the capabilities of the available assembly facilities and relative difficulty of each assembly task is used to make a comprehensive evaluation of assemblability.

Based on the sequence of motions generated using the method presented in this paper, every motion (or a set of motions) of a subassembly can be mapped to an assembly task. An assembly task represents a high level assembly process plan which results in the motions that map onto the task. A library of primitive assembly operations (referred to as subtasks) are defined. Each task consists of a set of subtask instantiations. For each task, the assemblability can be evaluated based on the facility model being developed. In the case the design is not assemblable, design modifications are suggested using the available facility model, or, additional facilities are selected without modifying the design. When the design can be assembled, the assembly task is automatically planned and programmed as a sequence

CH2872-0/90/0000-0399 © $1.00 1990 IEEE

of standardized subtasks.

This paper is organized as follows : Section 2 describes the model of the MSA. Section 3 describes the geometric Reasoning System-I which generates a set of assembly actions and an assembly sequence from the model. Algorithms for generating collision free translational and rotational motions are described. The reasoning of the assembly operations vis-a-vis the facilities (Reasoning System-II) has not been described in this paper. In Section 4 an example of an MSA is considered and the generation of assembly motions is illustrated. Section 5 summarizes this paper.

3. Representation of the MSA

In modeling MSAs for assembly analysis, the following attributes are of importance : Form, Material, Dimensions, Surface Quality, Tolerances, Geometric Features and Mating Conditions. Fundamental to most abstractions of mechanical assemblies is the geometry and topology of the various components and the mating between them. In our work, a 3-D geometric model of the MSA is created using the geometric modeler *Noodles*. This geometric model constitutes a representation in terms of low level geometric primitives (i.e., nodes, edges, faces and regions). The geometric representation is augmented by a topological framework (constructed within *Noodles*). Most of the higher level abstractions can be derived from this topology and geometry rich model, while some others, such as abstractions representing function and behavior, may require human input to augment the model.

Of primary importance in our effort to determine assembly operations are the geometric description and higher level abstractions relating to part mating and spatial occupation. Such higher level attributes are represented implicitly within the geometric modeler, although they involve varying degrees of complexity to derive from the geometric model. Sedas and Talukdar [12] used a stick model to represent spatial occupation and reason about the disassembly. This 2-D abstraction limits the utility of their approach to only symmetric MSAs and does not allow an easy extension to 3-D. In our work, inquiries about spatial occupation are made directly to the 3-D *Noodles* model. Mating relationships play an important role in our reasoning mechanism and are therefore represented explicitly in the form of a graph.

The internal representation of *Noodles* facilitates the generation of other abstractions of a MSA that are required for the reasoning system. *Noodles* provides a very powerful representation scheme for describing the geometry and topology of mechanical systems. Geometric models of individual components are created by the designer using functions provided by *Noodles*. The models of these individual components are then combined to create a model of the MSA. In the sequel, we present a very brief description of *Noodles*. A detailed report can be found in [3].

3.1. The Geometric Modeler *Noodles*

As described earlier, a model of the MSA is created using the geometric modeling system *Noodles*. This modeler uses a surface boundary representation scheme, i.e., it models objects by their enclosing shells. In particular, it employs a non-manifold scheme for surface boundary representation. The fundamental geometric elements (namely *nodes, edges, faces* and *regions*) are interpreted as point sets in R^3. The entire space is categorized into disjoint point sets, consisting of these four fundamental elements. Although these elements are disjoint, there is a relationship of immediate neighborhood among them. The geometric data is augmented with topological information using support elements. As a result of attempting to categorize space and explicitly represent their topology, *Noodles* demonstrates a significant improvement over contemporary geometric modelers in its receptiveness to interrogations from the Reasoning Systems I and II (in Figure 1) about the geometry and topology of assemblies.

3.2. Abstractions from the Geometric Model

The two main abstraction created are the *Component Graph* and the *Disassembly Graph.* Apart from the geometric model, an MSA is also represented at another level of abstraction. The Component Graph emphasizes the matings between components and subassemblies. The purpose of this abstraction is to have a knowledge representation more appropriate for the high level reasoning for assembly plan generation and task generation. The assembly procedure is represented in the form of an AND/OR graph (referred to as the Disassembly Graph) which could be thought of as another abstraction of the MSA. The geometric representation, however, forms the basis of these abstractions.

3.2.1. The Component Graph

The Component Graph is an undirected graph which represents the mating between the components in a subassembly. *Nodes* in the graph represent either individual components, subassemblies or void regions; *links* represent the mating conditions between the nodes. *Mating* consists of the faces that are shared by the two concerned nodes. One very useful addition has been made to this graph. We introduce a node that represents the region of space that surrounds the assembly. In the context of the graph, this special node has links to all components which are accessible from the outside. This information is valuable to the Reasoning System-I (Section 4). This graph is constructed automatically by making inquiries into the geometric model. Figure 2 shows a peg-in-hole assembly with the mating faces which form part of the link between the two nodes.

The purpose of the component graph is to give fast and easy access to information about the immediate neighborhood of each component in a given subassembly. Based on the mating conditions between a component and its neighbors, an evaluation of the constraints on the degrees of freedom of a component (or group of components) is made. This is used to recognize groups of components as being part of a meaningful subassembly.

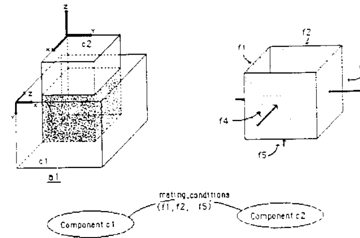

Figure 2: (a)A Peg-in-hole assembly. (b)The mating *faces*. (c) The *link* between c1 and c2.

Another merit in constructing this intermediate representation of part matings is that it makes the assembly task generation routines independent of the geometric modeling system.

During the reasoning process, when various subassemblies are being considered for potential candidates for disassembly, the component graph is modified to represent topologically connected groups of varying constituent components. Since a number of different subassembly representations are required, each for brief periods during the reasoning process, we have devised a two *tier* graph. There exists a base component graph that was created with each component as a node. When a subassembly with components $\{C_1, C_2, \ldots C_n\}$ needs to be created, the nodes in the base graph that represent the components $\{C_1, C_2, \ldots C_n\}$ are deactivated and a new node is created at the upper tier.

Given the model of any MSA, our objective is to determine a sequence of assembly tasks that would assemble the MSA. With a similar objective, Lee and Ko [6] developed a method for automated generation of an assembly procedure for an assembly. Their procedure requires the designer

to input the mating conditions and give qualitative descriptions of the kind of mating (namely against, fits, tight-fits and contact) between components. From the constructed graph they create a *hierarchy of components* based on the number and type of links that connect to each node. Their approach however does not involve any geometric considerations, such as accessibility and ease for grasping, stability, etc. to determine meaningful subassemblies. We believe that knowledge about the detailed geometry of assemblies allows us to automate the evaluation of assembly characteristics and behavior, without requiring detailed human input.

The central idea behind our method to generate assembly tasks is to **disassemble** the MSA. The disassembly procedure is part of the reasoning system and is described in the Section 4. The result of the disassembly procedure (and possible alternatives) is recorded in the *Disassembly Graph.* This graph represents the assembly tasks that would assemble the MSA.

3.2.2. The Disassembly Graph

The Reasoning System-I "disassembles" the MSA - it generates a set of disassembly tasks. The affect of applying a disassembly task/operation on a subassembly results in it being split into two smaller subassemblies. If this *splitting operation* is applied recursively to each emerging subassembly that consists of more than one component, we would have disassembled the MSA. The disassembly procedure is represented using an AND/OR tree like data structure, with the original MSA as the root, subassemblies as branch points and components as leaves, with disassembly operations represented on the interconnecting links. If similar subassemblies are generated, they are represented as the same node, which makes the AND/OR tree an AND/OR graph. If a given subassembly can be disassembled in more than one way, then an OR node is formed, and each of the immediate successors of the OR node represent AND nodes. Each AND node represents one way in which the subassembly can be disassembled.

The disassembly graph that is created, is similar to the one described in [5] to represent assembly sequences. Note that within this *task hierarchy*, there is an implicit hierarchy of subassemblies and components. Previous work, such as that by Lee [6], has attempted to generate a hierarchy of components, and then determine assembly operations based on this part hierarchy. Arranging components in an hierarchical order and determining assembly tasks from them is rather artificial. Imposing a hierarchy on sets of components is rather like working in reverse, as the hierarchical decomposition should emerge from the assembly process, and not the other way around.

An example of an MSA and its corresponding Disassembly Graph is shown in Figure 3. As can be seen, the root node represents the complete assembly of four components. The first disassembly operation corresponds to the unscrewing of component c4 from component c3. Since there is no alternative disassembly operation, the root node would be an AND node. In case some alternatives were found, the root node would be a OR node, with each of its child nodes representing each alternative. In this case, the root node gives rise to two child nodes, corresponding to the component c4 and the rest of the assembly (components c1, c2 and c3). Since node n1 represents a single component, it is a terminal node. Node n2 can now be treated as a new subassembly, and the splitting procedure is applied to node n2. This process is continued to generate the complete component hierarchy. In this case, since there is only one disassembly sequence, there are no OR branches in the tree, but in general they would be present.

The definition of an assembly task depends on part motions as well as the geometry of the components that are involved. Each assembly task can be defined in terms of a set of *primitive assembly operations* which are referred to as *subtasks.* The reasoning system-I determines feasible subassemblies and their motions that lead to disassembly. Based on the geometry and the motion of the moving subassemblies, a set of assembly tasks are identified. Kondelon [11] studied a number of products and their assembly methods and came up with a set of twelve *primitive* tasks associated with the assembly process. In our work we would like to define tasks at a lower level of breakdown; at a level that more appropriately reflects the exact manner in which these assembly tasks would be carried out. For example, when a sliding motion occurs, we classify it based on the number of degrees-of-freedom that the moving part possesses. Thus the case of sliding contact with respect to only one plane would be classified as a different task compared to sliding contact with respect to say two planes. The difference is that the moving body is constrained differently in the two cases; hence requiring different considerations during the actual assembly. When we attempt to reason about the assembly tasks vis-a-vis the assembly facilities, we must have available a detailed description of the assembly tasks.

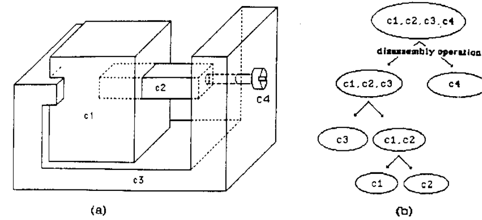

Figure 3: (a)Example of an assembly (b)Disassembly Graph

The next section describes the working of the Reasoning System-I : how the abstractions are used, how subassemblies are generated, how the disassembly algorithm works and how collision-free part motions are determined. A more detailed description of Reasoning System-I is presented in [10].

4. The Reasoning System-I

The reasoning system-I generates a sequence of assembly tasks using as input a model of an MSA and its component graph. De Fazio and Whitney [13] have described a method for generating all assembly sequences, based on the response of the user to a set of questions. The questions concern the *order* in which these assembly steps are performed during the assembly process. In comparison, although our goal is not to determine *all* assembly sequences, we adopt a more direct procedure - determine assembly operations and their sequence, without attempting to specify any precedence explicitly. Precedence relations fall out implicitly as a consequence of performing a *disassembly.* Sedas and Talukdar [12] have described a *disassembly expert* that imitates the human method of trial and error to simulate a disassembly process. In most cases, the sequence of assembly operations can be got by reversing the sequence of disassembly operations.

We follow similar lines, in that assembly operations required to assemble an MSA can be determined effectively by studying the way in which the MSA is disassembled. Each assembly task can be thought of as being some process by which the degrees of freedom of the components involved in that operation get constrained. An unassembled component has six degrees-of-freedom, 3 translational and 3 rotational. An assembly process has the effect of constraining some or all of these degrees-of-freedom. The output of this system is a Disassembly Graph accompanied by a graphic display of the disassembly process.

The Reasoning System-I is best explained in terms of modules. Module *Split* consists of three main parts: *Find_Subassembly, Determine_Motions* and *Identify_Tasks.* Given an assembly, this module 'splits' it into two subassemblies; those that are formed as a result of a disassembly operation. Module *Find_Subassembly* accepts a subassembly, consisting of say *n*

components, and finds a suitable division of these *n* components into two groups (or two subassemblies) such that one of them possess some *dof*. Once a suitable subassembly has been chosen, module *Determine_Motions* starts moving the subassembly out of the parent assembly. **Motions** that separate the subassembly from the rest of the MSA are determined. Based on the two subassemblies that are generated, and the motions that bring about a separation, module *Identify_Tasks* identifies the set of primitive assembly operations that generate the motions. It also defines the parameters that define this primitive operation uniquely.

An important part of this splitting procedure is the determination of collision free motion of the subassembly out of the parent assembly. Module *Determine_Motions* performs this function. In case no collision free motion can be determined, a backtracking mechanism is initiated such that other subassemblies can be generated by *Find_Subassembly*. Our method of determining collision free motions takes advantage of the nature of the problem at hand. Any motion is composed of primitive translational and rotational motions. This module outputs an ordered set of primitive motions which when applied to the MSA would disassemble it. Section 4.1 of this paper describes this module in detail.

The module *Identify_Tasks* takes as input the set of primitive motions and the description of the subassembly that is being disassembled. From these, it identifies "assembly tasks" that bring about these motions. As described earlier, a library of assembly subtasks is defined. A task represents an action on a subassembly. The definition of each task contains a list of primitive motions, each motion being generated by a subtask, which when made to act on a subassembly results in the subassembly being assembled. It also contains information on what kinds of geometric features need to be present on the subassembly if this task could act on it.

4.1. Generating Valid Disassembly Motions - Translational and Rotational

The degrees-of-freedom (*dof*) of nodes in a component graph can be inferred from the links that connects that node to other nodes. This information is used in identifying a suitable subassembly ($\mathbf{S}_1$) that can potentially be disassembled, as well as in proposing a direction of motion of the subassembly out of the parent assembly ($\mathbf{A}_1$). But this does not ensure a valid collision free path for the moving subassembly $\mathbf{S}_1$; a collision free path for $\mathbf{S}_1$ out of $\mathbf{A}_1$ must be found. In its most general formulation, this is thus a *find-path* problem.

Lozano-Perez et. al [7, 8, 9] have described the configuration space approach to solve this problem. Brooks [2] represents free space as generalized cones, and presents an algorithm to find collision free paths. However, these methods are fairly difficult to implement for 3-D objects that are translating and rotating. One very critical difference exists between the kinds of problems we are solving in this application and the general *find-path* problem, i.e., we are concerned with the motion of a subassembly within its parent assembly that has/had **constrained** this subassembly in a well defined manner, while the general *find-path* problem is to find a continuous path of a polyhedron between two positions in space in the presence of arbitrary polyhedral obstacles. Our problem of finding collision free motions has obstacles that are in very close proximity of the moving objects (in fact, they are usually in contact with the moving objects). It is this proximity that we use to our advantage.

Through a systematic study of the geometric constraints imposed by the stationary subassembly on the moving subassembly, we can formulate our problem differently. If looked upon as a search for a valid path, in this new premise, a *generate and test* approach does not involve extensive search. Moreover, by augmenting the search using knowledge of the geometric features that constrain the moving subassembly as well as those features present on (and surrounding) the moving subassembly, the search can be made fairly efficient. We pose ourselves the problem of reasoning about changes in *dof* of subassemblies due to translational and rotational motions. The heart of such a system requires collision detection capabilities between moving objects and stationary obstacles. We have developed efficient translational and rotational collision detection algorithms, which are described in the following paragraphs. These would apply to any surface boundary based geometric representation which approximates the object shape using planar facets.

4.1.1. Determining Translational Motion

Given a subassembly and the direction in which it can potentially be disassembled, this module detects whether motion of the subassembly in that direction will cause a collision with the rest of the assembly, and if so, the module determines the amount of allowable motion. The module uses a simple swept volume method, i.e., it generates a volume that is formed when the subassembly moves in the given direction and checks whether this volume intersects with any of the stationary components. Information about containment shells of subassemblies, directions of the normals of facets, bounding boxes, etc. is used to reduce the number of intersection calculations (face-face, face-edge and edge-edge). Since we possess a complete geometric model of the MSA, such information can be extracted at will. Once a collision is detected, the subassembly is moved to the point where the collision occurred and the process of finding new motions is continued.

4.1.2. Determining Rotational Motion

This section describes an approach to efficiently compute rotational *dof* within assemblies. Using this information, disassembly operations which involve rotational motions can be deduced. One class of subassemblies that potentially possess rotational *dof* are those that have a peg like cylindrical shape feature, which mates with a cylindrical hole like feature (we will refer to this as a cylindrical *peg-in-hole* type of mating). Thus, in order to check for rotational *dof*, peg-in-hole type of mating features would have to be identified. Currently we assume that such features can be identified using either help from the user or some other automatic feature recognition techniques. The axis of the cylindrical set of features is the *axis of rotation* of the subassembly.

Given an axis (**A**) in space and a subassembly ($\mathbf{S}_1$) that has a rotational *dof* about **A**, we need to determine the amount of rotation of $\mathbf{S}_1$ about **A**. The *amount* of rotation is determined by the state of the final position of $\mathbf{S}_1$: either it cannot rotate any more (in which case a collision has occurred), or its *dof* have changed to some desirable value. The actual rotation of $\mathbf{S}_1$ would result in it being completely disassembled, or, would lead to a *state* where some subassembly (not necessarily $\mathbf{S}_1$) has its *dof* modified. Most often, it is $\mathbf{S}_1$ that has one of its translational *dof* modified, which means that after the rotation is performed on $\mathbf{S}_1$ it can be translated to continue the disassembly process. This is what is meant by desirable value of *dof* of $\mathbf{S}_1$. A simple example is the twist-and-pullout action (eg. bayonet lock) which requires a rotation of the peg by a certain amount resulting in a state in which the peg in free to translate along its axis. The following paragraphs outline an efficient method of determining the amount of rotation that a subassembly can make about an axis before it collides with a part of the remaining assembly.

Outline of an Algorithm to efficiently determine Rotational Intersections

Consider an assembly, say of *n* components. Let $\{\mathbf{C}_s\}$ be a set of stationary components, and $\{\mathbf{C}_r\}$ a set of components that is being rotated about axis **A**. Note that $\{\mathbf{C}_r\}$ is the complement of $\{\mathbf{C}_s\}$ about the set of all components within the given assembly. The problem is to determine whether $\{\mathbf{C}_r\}$ collides with $\{\mathbf{C}_s\}$ during the course of its rotation about **A**, and if so, through what angle it rotates before collision occurs. The broad outline of the algorithm is as follows. The description of the model is transformed from the cartesian coordinate system to the cylindrical coordinate system (actually the transformation is performed only for faces

and edges that form part of the outer shell of $\{C_r\}$ and relevant elements of the shell of $\{C_s\}$). *Cylindrical bounding boxes* are created around such edges and faces. The volume traversed by $\{C_r\}$ during rotation is swept out by a subset of the edges and faces of $\{C_r\}$. The set $\{E_r\}$ of all such edges and faces $\{F_r\}$ are identified. Faces and edges that belong to $\{C_s\}$ and that form the neighborhood of the rotating subassembly are grouped into face set $\{F_s\}$ and edge set $\{E_s\}$. Each member of $\{E_r\}$ is checked for collision in a rotating frame with the members of $\{F_s\}$. In a similar fashion, each member of $\{E_s\}$ is tested with members of $\{F_r\}$, but with the rotation in the reverse direction. This final check is equivalent to keeping $\{C_r\}$ fixed and rotating $\{C_s\}$ in the opposite direction. This algorithm gives the amount of rotation that $\{C_r\}$ can make about the given axis before collision. This method of computing rotation collisions by using cylindrical coordinates transformations and intersection algorithms between rotating edges and stationary faces applies to any set of arbitrary polyhedral objects.

5. Example

The capabilities of the proposed disassembly method is demonstrated in this section using an example which has been taken from [4]. Figure 4 shows a series of snapshots during the disassembly of the MSA, as generated automatically by our program. The MSA consists of 4 parts : a box which has a slotted block fitting inside it, a door which fits onto a cylindrical bar which is on the box and acts like a hinge, and a key which passes through the slot in the block and gets locked in the slot. The program is supplied a model of the complete MSA (as in Figure 4(1)). The purpose in selecting this example is to highlight the generation of *translational and rotational motions* to disassemble an MSA.

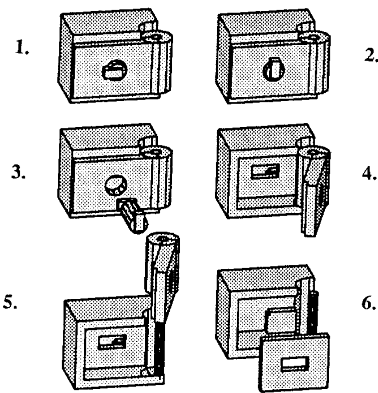

Figure 4: An example showing the disassembly process.

6. Summary

In this paper we have presented a method for automated generation of an assembly procedure for a given assembly. The procedure is generated from the parts geometry and topology model, and thus explicit specification of mating relations between parts is not required. The approach not only dispenses with the time consuming specification of mating relations by the user, but also prevents from wrong mating relations being presented, and makes unnecessary the need to check for validity. Moreover, the geometric considerations in the generation of the assembly sequence makes it possible to directly evaluate assembly performance and further to easily generate detailed assembly plans, such as grasping, transferring, and manipulation, based on the given geometry of the assembly.

We have presented algorithms to find a collision free path to yield disassembly motions. To disassemble an MSA we determine the parts or subassemblies that can separate themselves from the MSA after series of translational and rotational motions. The problem of deducing such motions is posed as one of determining changes in degree-of-freedom of subassemblies due to either translational or rotational motion. We have developed algorithms to determine locations at which collisions occur between moving subassemblies and the rest of the parent MSA. After the motions are deduced, a series of transformations is performed on the model of the MSA to accomplish disassembly. The efficiency and feasibility of the algorithms have been demonstrated through a case study. Our current work is addressing the modeling of facilities and automatic generation of downloadable code for executing on the assembly facility. The assembly facility that we plan to use initially is the TROIKABOT system developed by Westinghouse.

7. Acknowledgements

This research was supported in part by the Engineering Design Research Center, an NSF Engineering Research Center at Carnegie Mellon University.

References

1. G. Boothroyd, and P. Dewhurst. *Design for Assembly.* Boothroyd Dewhurst Inc., 1972.

2. R.A. Brooks. "Solving the Find-Path Problem by Good Representation of Free Space". *IEEE trans. on Systems, Man and Cybernetics SMC-13, No.3* (1983).

3. L. Gursoz., Y. Choi, and F. Prinz. Vertex-Based representation of Non-Manifold Boundaries. In *Geometric Modeling for Product Engineering*, North - Holland, New York, 1988, pp. 107 - 130.

4. R. Hoffman. Automated Assembly in a CSG Domain. Proc. IEEE Conference on Robotics and Automation, 1989, pp. 210-215.

5. L. Homem de Mello, and A. Sanderson. "AND/OR Graph Representation of Assembly Plans". *Technical Report, The Robotics Institute CMU-RI-TR-86-8* (1986).

6. K. Lee, and H. Ko. "Automatic Assembly Procedure Generation from Mating Conditions". *CAD 19*, 1 (1987), 3-10.

7. T. Lozano-Perez. "Automatic Planning of Manipulator Transfer Movements". *IEEE trans. on Systems, Man and Cybernetics SMC-11* (1981), 681-698.

8. T. Lozano-Perez. "Spatial Planning: A Configuration Space Approach". *IEEE trans. on Computers C-32, No.2* (1983), 108-120.

9. T. Lozano-Perez, and M.A. Wesley. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles". *Comm. of the ACM 22, No.10* (1979), 560-570.

10. R.S. Mattikalli, and P.K. Khosla. "A System to Determine Assembly Sequences". *Technical Report, Engineering Design Research Center, Carnegie Mellon University EDRC-24-16-89* (1989).

11. T. Owen. *Assembly with Robots.* Prentice Hall, 1982.

12. S. Sedas, and S. Talukdar. "Disassembly Expert". *Technical Report, Engineering Design Research Center, Carnegie Mellon University EDRC-01-03-87* (1987).

13. D. Whitney, and T. DeFazio. "Simplified Generation of all Mechanical Sequences". *IEEE Journal of Robotics and Automation RA-3*, 6 (December, 1987).