2002

An interactive approach of assembly planning

Xiaobu Yuan
University of Windsor

An Interactive Approach of Assembly Planning

Xiaobu Yuan[1]

[1]School of Computer Science, University of Windsor, Windsor, Ontario, Canada N9B 3P4

Abstract

Presented in this paper is an interactive approach of assembly planning. It provides a virtual reality interface for production engineers to program the virtual representation of robotic manipulators in a three-dimensional operation space. The direct human involvement creates a user-defined assembly sequence, which contains the human knowledge of mechanical assembly. By extracting the precedence relationship of machinery parts, for the first time it becomes possible to generate alternative assembly sequences automatically from a single sequence for robot reprogramming. This interactive approach introduces human expertise into assembly planning, thus breaking down the computational complexity of autonomous systems. Experiment and analysis provide strong evidence to support the incontestable advantages of "manufacturing in the computer".

1 Introduction

Given a set of machinery parts, the goal of assembly planning is to produce optimized sequences of assembly tasks for robotic manipulators to put together the product from its components in a cost-effective fashion. Due to the dominating effect that assembly operations constitute in industrial manufacturing, assembly planning has become one of the most expensive and time-consuming segments of production processes. Meanwhile, the strong influence that assembly planning imposes on the productivity of modern industry and on the study of robotics has inspired uninterrupted research in the past decades, and will continue to be an active topic in the future.

Assembly planning mainly consists of two processes, i.e., *robot programming* to teach robotic manipulators carrying out assembly tasks and *sequence planning* to determine the order of assembly tasks for optimal mechanical assembly. The old-fashion online approach suffers from the downtime of robot operations, the danger imposed upon human operators, and the difficulty of making adjustments for new products. Working offline inside the computer, in comparison, promotes the development of automated manufacturing tools and allows for the integration of different technologies from a wide range of originally separated areas.

Virtual assembly, for example, develops computer tools, and helps to make assembly-related engineering decisions through analysis, predictive models, visualization, and data presentation without realizing any of the physical products and their supporting processes [4]. Nevertheless, moving away from the physical world to the virtual world also creates new challenges. In particular, assembly planning faces a complexity problem whose solution relies on path searching in a high-dimensional configuration space [5].

This complexity problem exits in such systems that have to decide every movement of the machinery parts and all the joint values of robotic manipulators. In practice, it is common to reduce the complexity by making assumptions. Free-flying objects, for instance, simplify

the problem into a logical planning problem [9]. However, it is only a sub-problem of assembly planning as it leaves robotic execution out of assembly plans. Among the approaches that work on the complexity problem, the most influential are the numerical potential field, connectivity characterizing, and sequential frameworks [2].

In fact, human beings are excellent in thinking and working in the three-dimensional world. Researchers have already started to consider about introducing human expertise into assembly planning [3], but difficulties arise from the inconsistency in object manipulation between the physical and virtual worlds. Window-based human/computer interaction is not sufficient for three-dimensional operations. On the other hand, virtual realty allows human operators to work in a similar way as in the physical world. It is therefore particularly useful for the engagement of human expertise in the planning activities.

Applying virtual reality technology, this paper develops a technique to engage human operators in an online-style offline robot programming. It extracts the precedence relationship of machinery parts from user-defined assembly sequences, and constructs assembly trees to generate alternative assembly sequences for optimization and reprogramming. This approach of interactive assembly planning make it possible to integrate robot programming with sequence planning. It operates at a low complexity, and does not need the assumption of free-flying objects.

2 Virtual Programming

Robot programming teaches a robotic manipulator how to perform assembly tasks by specifying the manipulator's trajectory of motion and its end effector's functions. As shown in Fig. 1, the teaching pendant in a virtual environment is simply a set of graphics models, each of which represents a segment of the physical manipulator. The connection of segments forms a linkage of joints, and the numbering of joints goes from

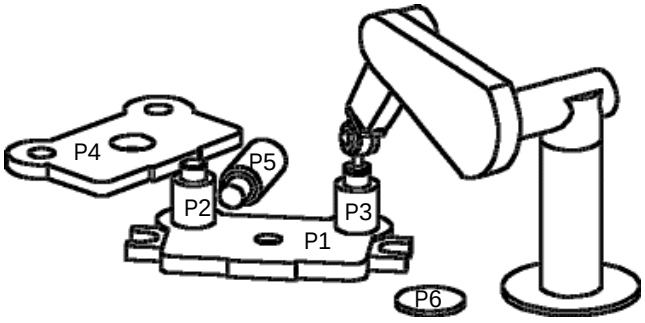

Figure 1: The Assembly of a Die-Set

the fixed base to the end effector. For any two adjacent joints, a homogeneous transformation $m_{i-1}^{i} = R_z(\theta_i)T_z(d_i)T_x(a_i)R_x(\alpha_{i-1})$ determines their relationship, $0 < i \leq l$, where l is the linkage's degree of freedom.

Every assembly task involves three stages of positioning [10]. The first stage specifies the preassembly position of an object. In virtual programming, it positions a virtual robotic manipulator and prepares its end effector for object handling. A human operator signals this position by grasping an object with his hand after touching the object. This operation ensures the human hand actually reaches to the object, which not only helps to physically bring the end effector of a robotic manipulator close to the object but also makes possible for the mapping of human grasping to the manipulator.

The second stage considers the in-assembly positions of an object. The practical nature of mechanical assembly requires the object to move along a safe and realizable path through the environment. The action that the human operator performs in the first stage prepares him for physical-level object manipulation in the second stage. The operator uses his hand to move the object from its starting position toward its targeted mating position, specifying the trajectory of the end effector. At the same time, inverse kinematics applied on the transformation sequence $m_{n-1}^{n} \cdots m_1^2 m_0^1$ computes the joint angles of the simulated robotic manipulator.

Let the initial set of joint angles for an l-link

robotic manipulator be $< j_1(t_0), \cdots j_l(t_0) >$ at the pre-assembly position. Inverse kinematics takes the hand position as the end effector's position at a given time t, and produces a new set of joint angles $< j_1(t), \cdots j_l(t) >$ according to the sampled tracking data. The result in turn refreshes the configuration of the virtual manipulator in display, which gives the operator visual feedback about the distance between objects. If the calculation has no solution, it means the object's position is unreachable. In the case when there are multiple solutions, the chosen solution is the one that presents the smallest difference of joint angles right before time t along the trajectory.

In the second stage of object positioning, the operator uses primarily his visual sense and his skills of working in the three-dimensional workspace to ensure a collision-free path of the object. The lack of physical restrictions in a virtual environment, however, does not permit accurate specification of fine motion. The last stage of object positioning in virtual programming therefore only needs the operator to bring the moving object to its destination with a close line-up of the reference-alignment coordinate systems along their major axes [8]. The operator then releases the object, and the system completes the desired alignment operation to establish the post-assembly position of the object.

Virtual programming allows the human operator to teach a robotic manipulator performing assembly operations through three-dimensional human/computer interaction. The procedure that the operator puts a set of machinery parts together into a product actually defines a sequence of assembly tasks. Suppose a product P consists of n component parts $P_i, 1 \leq i \leq n$, and the user-defined sequence is the kth sequence in a total of m possible sequences to produce P. The user-defined assembly sequence then takes a form in the following format.

$$P_0 \overset{\tau_1(k)}{\rightarrow} P_1(k) \cdots \overset{\tau_j(k)}{\rightarrow} P_j(k) \cdots \overset{\tau_n(k)}{\rightarrow} P_n(k) \quad (1)$$

where P_0 is a stable workstation, and $\mathcal{T} = \tau_1\tau_2\cdots\tau_n$ is an ordered set of assembly tasks.

3 Precedence Knowledge Extraction

Precedence constraints refer to the constraints on the order of assembly tasks. They originate from the common restrictions of mechanical assembly that ensure the validity of assembly sequences, including the geometric relationship of machinery parts, the stability of assembly operations, and the realizability of robotic manipulators. An assembly sequence is feasible only if the operation of all its assembly tasks satisfies these precedence constraints. The extraction of precedence knowledge in virtual assembly is a reasoning process that establishes the precedence relationship of all machinery parts according to the information contained in a single user-defined assembly sequence.

3.1 Constraints Detection

There are two types of constraints — direct constraints and implicit constraints. Let notation $\texttt{PC}(P_i, P_j)$ stand for the constraint that a machinery part P_i must be assembled before P_j is in place, $1 \leq i, j \leq n$ and $i \neq j$. It is a direct constraint if $\texttt{PC}(P_i, P_j)$ can only be determined by geometric reasoning. Otherwise, it is an implicit constraint when $\texttt{PC}(P_i, P_j)$ is inferable by constraint propagation. An implicit constraint $\texttt{PC}(i, j)$ can be deduced from $\texttt{PC}(i, \mu)$ and $\texttt{PC}(\mu, j)$, $1 \leq \mu \leq n$ and $\mu \neq i$ and $\mu \neq j$, when both $\texttt{PC}(i, \mu)$ and $\texttt{PC}(\mu, j)$ are available for reasoning.

Constraints deduction starts with a preprocessing that verifies the actual location of objects after assembling. If a mechanical part is involved in the composition of any sub-assembly, the product finished by the assembly sequence of (1) provides the information to update the final location with the sub-assembly. A transformation of the alignment coordinate system on the mating object is enough to fulfill the cal-

culation. Otherwise, its final resting location is the same as the position after it is released from the hand for alignment. In addition, the gap between the releasing and alignment positions creates a sweeping volume, which identifies geometric constraints.

A direct constraint $\texttt{PC}(P_i, P_j)$ exists in two situations. The first situation arises when there is a direct reference-alignment relationship between P_j and P_i. It means that the reference coordinate system is on P_i and the alignment coordinate system is on P_j by specification or through revision in preprocessing. The first case takes care of normal placement of objects. In comparison, the second situation occurs when an intersection relationship exists between P_j and the sweeping volume of P_i. It handles implied geometric relationships.

For example, the assembly of a die-set as shown in Fig. 1 requires defining a pair of coordinate systems on P_2 and P_1 respectively to stand up P_2 at a guide-hole of P_1. This reference-alignment relationship makes up a direct constraint $\texttt{PC}(P_1, P_2)$. Another direct constraint $\texttt{PC}(P_3, P_4)$ is detected, however, mainly because there is no collision-free path to insert P_3 after P_4 is in place. It is verifiable by first placing P_4 on top of P_2, and then checking the geometric intersection between the sweeping volume of P_3 and the graphics model of P_4.

The die-set example also presents plenty of situations that lead to implicit constraints. For instance, the direct constraints $\texttt{PC}(P_1, P_2)$ and $\texttt{PC}(P_2, P_4)$ lead to an implicit constraint $\texttt{PC}(P_1, P_4)$, i.e., the assembly of the punch holder should follow the die holder; and $\texttt{PC}(P_1, P_4)$ and $\texttt{PC}(P_4, P_5)$ lead to $\texttt{PC}(P_1, P_5)$, which points out that the assembly of the punch should be done after the die holder is in place. Since the detection of implicit constraints involves with symbolic reasoning only, it is much faster to compute than the detection of direct constraints.

3.2 Constraints Deduction

In corresponding to the two different types of precedence constraints in assembling process, constraints deduction employs an $n \times n$ matrix M to maintain the result of both types of reasoning. All the values of elements $m(i, j)$ in the deduction matrix M are set to '*-9*', $0 \leq i \leq n-1$ and $1 \leq j \leq n$, at initialization. Both i and j are arranged in the order of machinery parts as they appear in the user-defined assembly sequence. The index along the columns is the actual index in the sequence, i.e., from 1 to n; but the index along the rows begins with 0 to include the workstation P_0 and ends with $n-1$.

Constraints deduction first uses geometric reasoning to decide direct constraints on the diagonal elements of M. For $m(i, j)$, $i = 0,1,\cdots,n-1$ and $j = i+1$, its value changes from '*-9*' to '*1*' if $\texttt{PC}(P_i, P_j)$ exists. Otherwise, there is no precedence constraint between P_i and P_j, and the value of $m(i, j)$ is reassigned to '*0*'. The deduction then propagates to the upper-right region of M. Additional geometric checks for direct constraints are necessary only if constraint propagation cannot determine the value of an element in the region.

In the upper-right region of M, constraints deduction propagates with i increasing one by one from '*0*' to $n-2$ and j from $i+1$ to $n-1$. An implicit constraint $\texttt{PC}(P_i, P_j)$ exists under the condition that both $\texttt{PC}(P_i, P_{i+1})$ and $\texttt{PC}(P_{i+1}, P_j)$ exist, i.e., the values of $m(i, i+1)$ and $m(i+1, j)$ are either '*1*' or '*-1*'. In such a case, $m(i, j)$ takes '*-1*' as its new value. Alternatively, geometric reasoning is in action again to decide for $\texttt{PC}(P_i, P_j)$. The value of $m(i, j)$ resets to '*1*' when confirmed, or '*0*' otherwise. The propagation continues until the last element in the upper-right region.

This constraint deduction encourages the use of symbolic reasoning whenever possible, thus cutting down the number of geometric reasong. At the end of this process, all elements along the diagonal and in the upper-right region of the deduction matrix are reset from '*-9*' to '*1*', '*0*', or

```
1.   Create a graph G with a level_0 node P_0.
2.   Create an empty set N_0(0).
3.   Set both node index i and level index k to 0;
4.   Link_Node(i, k)  {
5.      For j = i to n − 1 with j++,
6.        if m(i, j) = 1,
7.          if m(l, j)! = 1 for every l in N_i(k),
8.            create a level_{k+1} node P_j;
9.            make a level_k link from P_i to P_j;
10.           add j to N_i(k);
11.     For every j in N_i(k),
12.       create an empty set N_j(k + 1);
13.       add all the rest of N_i(k) to N_j(k + 1);
14.       Link_Node(j, k + 1) }.
```

Figure 2: An Algorithm of Tree Construction from Deduction Matrix

'*-1*'. Among them, an element $m(i,j)$ of a value '*1*' or '*-1*', $0 \le i < n-1$ and $i \le j < n$, tells that the assembly of P_i must be completed before the assembly of P_j. Due to the implied relationship between P_j and P_i in $m(i,j)$, there is no need to process the lower-left region of M.

4 Sequence Generation

Assmbly planning finally generates assembly sequences for optimization. The user-defined assembly sequence embodies knowledge of mechanical assembly, leading to the extraction of precedence constraints. These constraints influence the order of assembly tasks, and thus constitute the basics of sequence generation. With a further conversion from the deduction matrix to an assembly tree [11], the interactive approach of assembly planning develops an incomparable feature of virtual assembly — automatic generation of alternative assembly sequences from a single sequence.

An assembly tree is a simple form of directed graph for the representation assembly sequences [1]. In an assembly tree, a path from the root node to a leaf node defines an assembly sequence. The assembly tree for an $n \times n$ deduction matrix M has $n+1$ levels. At its first level, line 1 of the conversion algorithm in Fig. 2 creates a single node P_0 to indicate the workstation. The algorithm then prepares in line 2 an empty set for the node in the current level, and sets up in line 3 two indices i and k. The actual tree construction takes place in the recursive procedure **Link_Node**$(\cdot)$ whose operation consists of two blocks of actions.

The first block starts from line 5. It goes through the elements along the i-th row in the upper-right region of the deduction matrix, and checks for direct constraints. If $m(i,j) = 1$, P_j becomes a child node of P_i at the $(k+1)$-th level of the tree unless P_j is already in the set of $N_i(k)$. The first block of actions finishes by drawing a link from P_i to P_j and adding index j to $N_i(k)$ in line 9 and line 10 respectively. Following it in the remaining four lines from 11 to 14, the second block immediately creates an empty set $N_j(k+1)$ for every node in the current level. It also copies the other elements of $N_i(k)$ into $N_j(k+1)$, and initiates another round of process for them.

In such a way, the recursive procedure expands a node in the tree by taking as its child nodes all its sibling nodes and the others that tend to link to the node but not its siblings. At its completion, the assembly tree represents different ways of putting together a product with its components, including the one defined by the user. Any path connecting the root to a leaf node in the tree defines an assembly sequence as below. A different index k_1 indicates a different order of machinery parts $P_j(k_1)$, $j = 1, \cdots, n$, which means that $P_j(k_1)$ is not necessarily the same as $P_j(k_2)$ even with the same index j in two sequences.

$$P_0 \overset{\tau_1(k')}{\rightarrow} P_1(k') \cdots \overset{\tau_j(k')}{\rightarrow} P_j(k') \cdots \overset{\tau_n(k')}{\rightarrow} P_n(k') \quad (2)$$

It may be necessary for assembly planning to specify some of the assembly tasks once again

when evaluating the different assembly sequences against certain pre-selected criteria, such as the length and linearity of assembly sequences [9]. The main reason is that assembly tasks applied on the same objects are presumably the same in all sequences as those interactively defined in the user-defined assembly sequence. Due to the change of order in the alternative sequences, some tasks may need adjustments. Whenever necessary, interactive assembly planning plays back all the selected sequences and engages the operator back into the virtual environment for him to redefine tasks.

5 Experiments and Discussion

Experiments were conducted to examine the performance of interactive assembly planning via virtual reality. The tested machinery sets covered different object shapes and different task difficulties. The emphasis, however, was on the operation of the proposed interactive approach and the new feature of sequences generation from a single user-defined assembly sequence.

5.1 An Experiment on the Die-Set

The typical die-set shown in Fig. 1 consists of three principal components: a punch holder (P_4), a die holder (P_1), and two guideposts (P_2 and P_3). It also has a stamping punch (P_5) and a metal plate (P_6) for coin-making. The equation in (3) gives a feasible sequence of assembling the die-set from its five components and placing the metal plate for stamping.

$$Tab \xrightarrow{\tau_1} P_1 \xrightarrow{\tau_2} P_2 \xrightarrow{\tau_3} P_3 \xrightarrow{\tau_4} P_4 \xrightarrow{\tau_5} P_5 \xrightarrow{\tau_6} P_6 \quad (3)$$

In the sequence, the assembly tasks applied upon the parts are as the following, where each of the tasks τ_k, $0 < k \leq 6$, includes the details to instruct a robotic manipulator reaching to, moving around, and releasing an object P_k.

M	P_1	P_2	P_3	P_4	P_5	P_6
P_0	1	-1	-1	-1	-1	-1
P_1	-9	1	1	-1	-1	1
P_2	-9	-9	0	1	-1	0
P_3	-9	-9	-9	1	-1	0
P_4	-9	-9	-9	-9	1	0
P_5	-9	-9	-9	-9	-9	0

Table 1: The Deduction Matrix of Die-Set

- τ_1: place P_1 with its half-open hole facing up;
- τ_2: stand up P_2 by inserting it into a guidehole;
- τ_3: stand up P_3 at the other guidehole of P_1;
- τ_4: sit the two corner-holes of P_4 on the guideposts;
- τ_5: insert P_5 half-way through the bigger hole of P_4;
- τ_6: slide P_6 on top of the half-open hole of P_1.

Right after the human operator finishes defining the assembly sequence of (3) in virtual programming, interactive assembly planning takes place to determine the precedence relationship between objects. Following the order of object manipulation, it checks all parts one by one for geometric constraints. The first object P_1 is always on the table P_0, therefore $m(0,0)$ in Table 1 is *1*. As P_2 is in a hole of P_1 through alignment, the *1*-labeled $m(1,1)$ indicates that τ_2 can take place only after τ_1. For a similar reason, the values of both $m(3,3)$ and $m(4,4)$ are *1*.

However, there is no reference-alignment relationship between either the pair of P_2 and P_3 or the pair of P_5 and P_6. The values of $m(2,2)$ and $m(5,5)$ have to come from the intersection check between P_3 or P_6 and the sweeping volume of P_2 or P_5. The result is *0* for both of them. For the elements in the upper-right region, symbolic reasoning takes place first. For example, the value of $m(0,1)$ becomes *-1* simply because $m(0,1)$ and $m(1,1)$ are already labeled with *1*. In the case of a *0*-labeled element in the inference, a check of geometric intersection is unavoidable. This is how $m(1,2)$ obtains its value

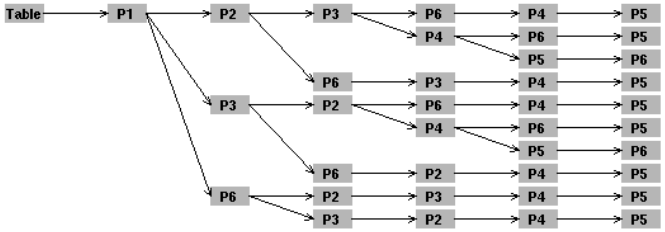

Figure 3: The Assembly Tree of the Die-Set

as $m(2,2)$ has a *0* value.

Sequence generation begins to construct the assembly tree after constraints deduction completes the deduction matrix of Table 1. The top two levels of the tree are a graph description of the first row of the matrix that links P_0 with all the objects with a *1*-labeled element in this row. Therefore, element $m(0,0)$ results in a link from P_0 to P_1 in the tree. The tree then expands node P_1 by adding its child nodes P_2, P_3, and P_6. The next level then expands the three nodes with their brothers as the algorithm of Fig. 2 forbids P_4 becoming a child because it has two parents P_2 and P_3 who happen to be brothers. The same rules apply on the rest of the tree, and the final product is a tree in Fig. 3.

Fig. 3 shows ten possible sequences to assemble the die-set, including the one defined by the operator. Taking a sequence out of the tree, a replay moves each object in the sequence one after another by applying the corresponding assembly task recorded during programming. For the first sequence in the tree, as an example, the operations are as below. Interactive evaluation shows that there is no problem applying the original tasks on this new sequence.

$$Tab \xrightarrow{\tau_1} P_1 \xrightarrow{\tau_2} P_2 \xrightarrow{\tau_3} P_3 \xrightarrow{\tau_6} P_6 \xrightarrow{\tau_4} P_4 \xrightarrow{\tau_5} P_5 \qquad (4)$$

5.2 Analysis and Discussions

Direct object manipulation in a three-dimensional operation space is the key to introduce human expertise into assembly planning. Through virtual reality, human operators use their hands in human/computer interaction

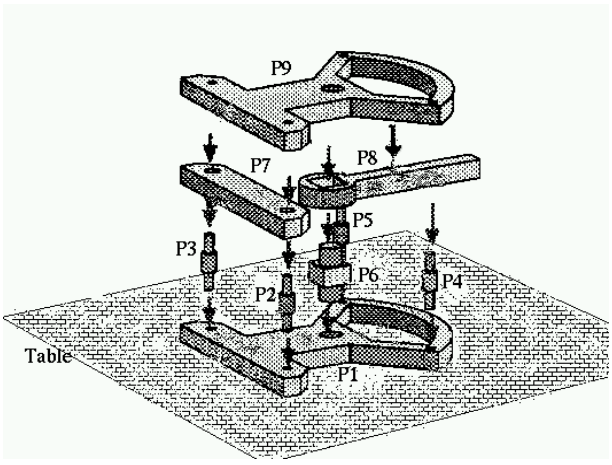

Figure 4: An Experiment on the Pendulum Assembly

in a similar way as in the physical world. In addition, direct object manipulation with hands allows the mapping of human grasping to the robotic manipulator. The typical role of a hand-based user interface, nevertheless, is in motion definition. With the help of inverse kinematics, hand movements contribute directly to the specification of assembly operations.

Other techniques are also available to improve the efficiency and convenience of user interfaces. Voice recognition, in particular, has joined forces with vision-based gesture recognition, which created an impressive new type of user interface. Recent advances have produced moderate success in applications that involves two-dimensional motions to, for example, control mobile robots. Computer vision, however, has not reached the maturity of replacing data glove devices to produce the detailed three-dimensional robotic instructions as required by assembly planning.

Interactive assembly planning includes both constraints deduction and sequence generations. For the elements along the diagonal and in the upper-right region of a deduction matrix, there is a need of $n!$ checks to establish precedence relationships. Experiment results indicate that a large portion of the checks uses symbolic rea-

S	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9
tab	1	-1	-1	-1	-1	-1	-1	-1	-1
P_1	-9	1	1	1	1	1	-1	-1	-1
P_2	-9	-9	0	0	0	0	1	0	-1
P_3	-9	-9	-9	0	0	0	1	0	-1
P_4	-9	-9	-9	-9	0	0	0	0	1
P_5	-9	-9	-9	-9	-9	0	0	0	1
P_6	-9	-9	-9	-9	-9	-9	0	1	-1
P_7	-9	-9	-9	-9	-9	-9	-9	0	1
P_8	-9	-9	-9	-9	-9	-9	-9	-9	1

Table 2: The Deduction Matrix for the Pendulum Assembly

soning. The remaining checks use reference-alignment relationship first and geometric intersection the last. As for tree construction, it is pure symbolic. Its complexity ranges from $O(n)$ to $O(n!n)$ depending on the product and its components.

Running the procedure on the classical Pendulum Assembly produced the constraint matrix in Table 2. The constraint deduction process conducted only *30* geometric checks. From the matrix, a total of *840* different feasible assembly sequences was then generated. When running the procedure on other products of different number and configuration of machinery parts, the worst case of sequence planning happens when no parts rely on each other in assembly. It is a typical example of combination explosion in the order of parts' number. The best case, on the other hand, happens when every part relies on one and only one of the other parts in assembly, in which only one sequence is possible.

6 Conclusion

Virtual assembly is a pilot project of a much bigger vision on "manufacturing in the computer". The interactive approach presented in this paper creates a way of introducing human expertise into assembly planning and a mechanism of integrating robot programming with sequence planning. Virtual assembly helps to identify and resolve issues related to the construction of an integrated virtual manufacturing environment that could enhance all levels of manufacturing decision and control.

References

[1] J. Bander. A heuristic-search algorithm for path determination with learning. *IEEE Transactions on Systems, Man, and Cybernetics, PART A: Systems and Humans*, 28(1):131–134, Jan 1998.

[2] K. Gupta. The sequential framework for practical motion planning for manipulator arms: Algorithm and experiments. In K. Gupta and A. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 9–31. John Wiley & Sons Ltd, 1998.

[3] K. Gupta and A. del Pobil. Successes, failures, challenges and future directions of robot motion planning. In K. Gupta and A. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 351–354. John Wiley & Sons Ltd, 1998.

[4] S. Jayaram, H. Connacher, and K. Lyons. Virtual assembly using virtual-reality techniques. *Computer-Aided Design*, 8(29):575–584, Aug 1997.

[5] J. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[6] S. Mok, K. Ong, and C. Wu. Automatic generation of assembly instructions using STEP. In *Proc. of 2001 IEEE International Conference on Robotics and Automation*, pages 313–318, May 2001.

[7] S. Mok, C. Wu, and D. Lee. Modeling automatic assembly and disassembly operations for virtual manufacturing. *IEEE Transactions on Systems, Man, and Cybernetics,*

PART A: Systems and Humans, 31(3):223–232, May 2001.

[8] H. Sun, X. Yuan, G. Baciu, and Y. Gu. Direct virtual-hand interface in robot assembly programming. *Journal of Visual Languages and Computing*, 10(1):55–68, 1999.

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

[10] C. Wu and N. Kim. Modeling of part-mating strategies for automating assembly operations for robots. *IEEE Transactions on Systems Man and Cybernetics*, 24(7):1065–1074, July 1994.

[11] X. Yuan. Interactive assembly planning in virtual environments. In *Proc. of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1462–1467, Nov. 2000.