Journal of Intelligent Manufacturing (1990) **1**, 17–29

An expert system for design for robotic assembly

CHIDAMBARAM MANOHARAN, HSU-PIN (BEN) WANG* and ANDRES SOOM

Department of Industrial Engineering, State University of New York at Buffalo, 342 Lawrence D. Bell Hall, Buffalo, NY 14260, USA

In recent years, due to the various advantages associated with automation and robotics, much work has been done in developing robotic systems for assembly operations. Since part design plays a major role in assembly, this paper deals with the design of parts for ease of robotic assembly. Considerable knowledge is available in the form of design for robotic assembly rules. In addition, a large amount of data is required for decisions regarding suitability of parts for robotic assembly. The implementation of design for robotic assembly rules would be much easier with the help of an expert system, which would guide the designer toward choosing the design alternative that can best facilitate ease of assembly from a robotic point of view.

To this end, a prototype expert system for design for robotic assembly is developed and presented in this paper. The expert system was implemented as a production system, which consists of rules and Object–Attribute–Value (O–A–V) triplets to represent domain knowledge. In order to best utilize the domain specific knowledge, a state space search-based inference mechanism was employed. The implementation of the prototype system is illustrated with examples.

Keywords: Robotic assembly, expert systems

1. Introduction

Product assembly has been a major area of interest in manufacturing industry. According to Riley (1982), 'For most American companies, assembly accounts for over half of all the direct costs involved in manufacturing.' Because of overseas competition and the continually increasing cost of manual labor, the need to increase productivity is of extreme importance for many American companies to remain competitive. Since assembly is such an important part of manufacturing, it is desirable to optimize the assembly process wherever possible.

A product design determines its structure, method of assembly, number of components, components design, tolerances, etc. Assembly is thus fundamentally established during the design stage. Unfortunately, the designer has traditionally taken little notice of assembly, but been preoccupied with achieving the desired product function.

A product is an assemblage of several number of parts. One reason a product may have to be divided into multiple components is the need for components to have degrees of freedom with respect to each other. Most products require relative motion between parts to perform their required function. For example, a piston in a cylinder must move relative to the sleeve to perform its required function. Often products must be separated into components because different materials are required. For example, many electrical components require a conductor and an insulator to achieve their functional requirements.

According to Boothroyd and Dewhurst (1983), the cost of assembling a product is related to both the design of the product and the assembly method used for its production. The lowest cost can be achieved by designing the product so that it can be economically assembled by the most appropriate assembly method. According to Payne,

> Mechanical assembly is so broad that it is difficult to provide a general solution for various applications. It involves many more variables than electronic assembly, which typically is a matter of mounting components on a flat surface. In the electronics field, it is possible to develop some engineering approaches that can be applied in a variety of applications and by different users. The part presentation and orientation problem is probably a tougher challenge in mechanical assembly systems.

Since assembly is a highly repetitive job, manual assembly tends to become tedious, tiring, monotonous and, over a period, inconsistent too, while on the other hand, robotic assembly has its own benefits: (1) consistency, (2) high

*To whom correspondence should be addressed.

positioning repeatability, and (3) improved product design.

Unfortunately, a majority of the products that are assembled manually, cannot be assembled robotically. The human hand, which has approximately 22 degrees of freedom (Wood, 1985), is an ideal gripper and manipulator for assembly operations. Moreover, human beings have sensory capabilities (e.g. vision, force feedback and temperature sensing) to aid them in the assembly process. A robotic assembly machine, on the other hand, is highly limited in its capabilities. A typical robot has between 3 and 6 degrees of freedom and limited mechanical force detection and vision capabilities. Therefore, it is imperative to take into consideration the capabilities of the robotic assembly system while developing a product design.

1.1. *Related work in design for robotic assembly*

For many years, researchers in this field have tried to aid the designer and traditionally this has taken the form of a few general design rules to help in the two major areas of assembly: feeding and orienting components and construction of assemblies. It is argued that these few general rules are likely to augment a designer's experience sufficiently to facilitate satisfactory design for automatic assembly. This view is reinforced through a survey reported by Schraft and Bassler (1984). It is clear that in order for the designer to solve the assembly problem, the designer needs access to a higher level of automatic knowhow than that available in these rules.

The most productive and best known work in this field is that conducted by Boothroyd and Dewhurst (1983), who are considered the forefathers of design for assembly. The 'Design for Assembly' system available in the form of a handbook and software, allows designers and manufacturing engineers to rate products or designs for ease of assembly and obtaining estimates of assembly costs.

In practice, design for robotic assembly begins with the coding of the components in a product for handling and assembly process, and then if the coding indicates areas of difficulty, an attempt is made to redesign the components. However, while the designer can be trained to use the coding system satisfactorily, the large amount of coding and form filling can be tedious and time consuming even to an experienced analyst. For example, the analysis of the eight component assembly discussed by Boothroyd and Redford (1968) required over 100 entries on the worksheet.

Andreasen and colleagues (1983) propose methods to design components to simplify assembly operations while maintaining required part functions. While their work does a very good job of illustrating a variety of design principles, it does not present a cohesive methodology.

The use of interactive computer programs for the analysis of components for manual and automatic assembly has been reported by Field and Russel (1980) and Boothroyd and Dewhurst (1983). In this work, computer programs have been developed to semi-automate some aspects of the analysis procedure in the *Design for Assembly* handbook. The programs are 'hard-wired' to pose a series of questions to obtain the data required to perform the design analysis. Graphical displays were used to aid the designer in coding the components (Dewhurst and Boothroyd, 1983).

A new methodology was developed by Wood (1985) to aid the design process from the initial conceptual stage. This system is more suitable for the synthesis of a new product rather than for analysis of a product that already exists.

Most of the motivation for this work was drawn from the work done by Becker (1986) and Swift (1987) and extends to some of the concepts given therein. The ASSEX (ASSembly design EXpert) system has some shortcomings. A large amount of data input is required for the reasoning. The system should be capable of interacting effectively with the other components of a Computer Integrated Manufacturing System, like the design department, manufacturing department, etc. Also, there is still doubt over how knowledge about the parts is going to be represented.

1.2. *Research objectives*

Based on the above-mentioned problems, the objectives of this study are the following:

(1) Study the feasibility of developing an expert system to facilitate the design for robotic assembly.

(2) Develop an efficient way of representing the known knowledge.

(3) Develop a control strategy or inference mechanism to work with the facts and rules.

(4) Reduce the amount of data input by the user by trying to elicit geometric data about the parts directly from a CAD Post Processor file in conjunction with the KK3 code [5].

(5) Develop a prototype expert system for design for robotic assembly.

This paper is organized as follows. Section 2 reviews the methodology and the framework of the proposed system. Section 3 discusses about the actual implementation of the proposed Expert System along with an associated example. The implementation of the knowledge and inference engine is dealt with in detail. Section 4 concludes this study and points out some directions for future research.

2. Methodology and framework of the system

The main goal for the design for assembly, is to reduce the assembly costs of the product that is being considered for

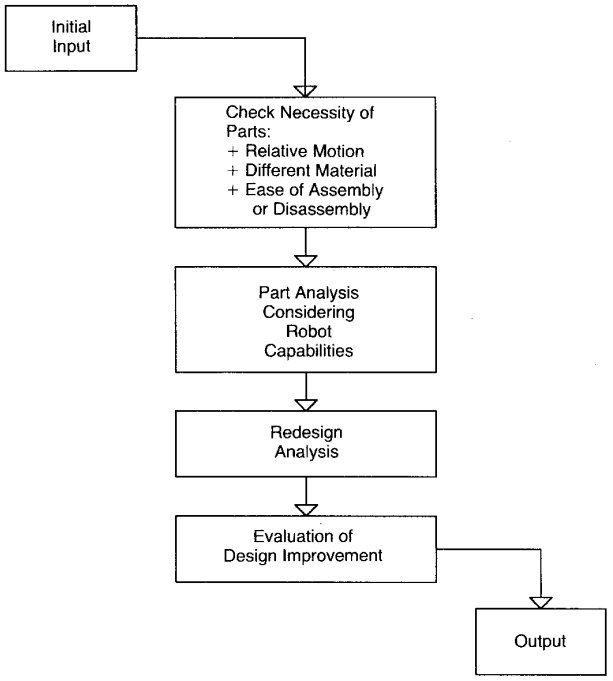

Fig. 1. Flow chart of redesign analysis.

redesign. This can be achieved by reducing the number of assembly operations and facilitating the assembly operations by designing for ease of orientation, manipulation, alignment and insertion.

Before attempting to develop an expert system, the procedure leading to a solution must be found. To aid this, the work of a human expert should be analyzed and his approach to finding a solution examined. This knowledge acquisition is the first and the most important step in developing an expert system. This knowledge about the domain can be elicited from experts in the field and the knowledge compiled in books, articles and other references.

After discovering the method for finding a solution, the specification for the expert system must be compiled. The domain-specific knowledge must be represented in an efficient way. This presentation must store the rules and facts about the system. Once the knowledge is organized, an efficient mechanism to deal with the rules and facts has to be developed.

The methodology of the system is shown as a flow chart in Fig. 1 and is described below.

(1) Since this system is based on analysis and not for the purposes of synthesis, the basic input to the system is an already designed product which has to be assembled by a robot. The initial input to the system includes a part list, mating relationships, a CAD file describing the surfaces and features of the parts, assembly sequence and other related information.

(2) While designing the parts, the capability of the assembly robot should also be taken into account. Since the capabilities may differ between robots, parts that can be assembled on one robot may not be suitable for another. Features of the robot such as accuracy, payload capacity, configuration, degree of freedom, gripper, etc. should be considered in redesign analysis.

(3) During analysis, the expert commences by making hypotheses about improvements. The first consideration would be to figure out the necessity of a particular part in the product or to establish its function. As mentioned earlier, the component in question should serve one of the following purposes:

(a) Relative motion.
(b) Material difference.
(c) Providing access for assembly or disassembly of parts already assembled.

If not, this part is an ideal candidate for elimination. For instance, the third and fourth digits of KK3 code provide detailed material information and can serve as a basis for part necessity checking. In this section of the analysis, only main components are specified. Fastening elements are considered in a later stage.

(4) The aim of this step is to ascertain how the parts are assembled and held in place. The capability of the robotic gripper will have an impact on some of the design features of the part.

(5) After this, analysis is carried out with respect to other criteria such as ease of orienting, manipulating, inserting and alignment that could be avoided by redesigning the parts.

(6) The interface between the components should also be taken into account while designing parts for assembly. This interface relationship along with the accuracy for locating parts with each other represents input by the user. Very fine accuracy requirements indicate potential positioning problems, which can typically be resolved by providing generous chamfers, tapers, etc.

(7) At this stage, the actual fastening method is defined for all the parts. It is desirable to design parts that are self-fastening (e.g. compliant tabs or snap-fit parts) and do not require additional fastening elements.

(8) The final step is to evaluate improvement of design in term of design efficiency. An estimate of the savings in the average assembly cycle time to produce the entire product is also provided along with the design efficiency.

2.1. *Part description*

The main focus of this system is a product and the parts that make up the product. For a redesign analysis, a comprehensive description about current design is necessary. The

product in its entirety has to be described to the system or in other words, the system has to 'visualize' the parts. In most other systems, the user has to input a voluminous amount of data to describe the product to the system.

In keeping up with the objectives of this study, an attempt has been made to reduce the amount of data input by the user. The ideal situation would be to do away completely with user interaction and directly interact with a CAD system. But technology at this point has not reached that degree of sophistication. To this effect, two methods have been employed to elicit geometric data directly from the KK3 code in conjunction with the output from a CAD post processor.

2.1.1. *KK3 code*

KK3 code is 21 digits long and spans the entire gamut of machined parts (Chang and Wysk, 1985). The KK3 code provides valuable information for design for assembly analysis:

(1) The main part shape is critical to the part orienting and feeding. Common part shape (bracket, cylinder, pin, etc.) information is available from KK3 code.

(2) Individual part features also play a vital role in part orienting and feeding analysis. These features are inferred from KK3 code.

(3) A KK3 code contains part accuracy information which is necessary for design for assembly analysis.

(4) Material information, which is available from a KK3 code, is often used in checking part necessity.

2.1.2. *CAD post processor*

Because of its importance in CAD/CAM integration, the extraction of geometric features from a CAD data base has received considerable attention since the mid 1970s. Wang has attempted to extract complete geometric information for symmetric rotational parts by recognizing all surface features (Wang and Chang, 1988). In this research, the authors make use of the output produced from the post processor of AIMSI (An Intelligent Machined Surfaces Identifier) (Wang, 1986). The format of the output file from the post processor is so designed that it includes most information necessary for later processing. Each drawing entity in the file occupies one record space and each record consists of nine data fields (see Table 1).

Surface type includes external and internal longitudinal cylinders, external and internal vertical faces, taper, convex and concave arcs and thread (Wang, 1986). Thus, given a part drawing and its related database (KK3 code, CAD postprocessor output, etc.) the main task is to identify main part shape and the presence of certain part attributes which are necessary to provide redesign suggestions.

Table 1. Data format of CAD postprocessor output.

Field	Description	Data Type
1	surface number	integer
2	surface type	integer
3	X_start_pt	real
4	Y_start_pt	real
5	X_end_pt	real
6	Y_end_pt	real
7	X_center_pt	real
8	Y_center_pt	real
9	radius	real

2.2 *State space search*

In order to provide a formal description of a problem, the following points have to be taken into consideration (Rich, 1986):

(1) Define a state space that contains all the possible configurations of the relevant objects.

(2) Specify one state as the *initial* state, that describes possible situations from which the problem-solving process may start.

(3) Specify one or more states as *final* states that would be acceptable as solutions to the problem.

(4) Provide a set of rules that describe the actions available for moving from the *initial* to the *final* states.

The problem can then be stated by using the rules in combination with an appropriate control strategy for controlling rule firing to move through the problem space until a goal (final) state is found.

Every search process can be viewed as a traversal of a directed graph, in which each node represents a problem state and each arc represents a relationship rule between the states represented by the nodes it connects. This is shown in Fig. 2, where S and F represent the initial and the final states of the system, $R1,R2,R3,\ldots Rn$ are the connecting links in the state diagram and $S1,S2,\ldots Sn$ are the intermediate states of the decision-making process. This graph that must be searched could, in principle, be constructed in its entirety from the rules that define allowable moves in the problem space, but in practice it is never done, since it is too large and most of it is never explored.

Instead of first building the graph explicitly and then searching it, most search programs represent the graph inplicitly in the rules and generate explicitly only those parts that they wish to explore. It is important to keep in mind this distinction between implicit search graphs and

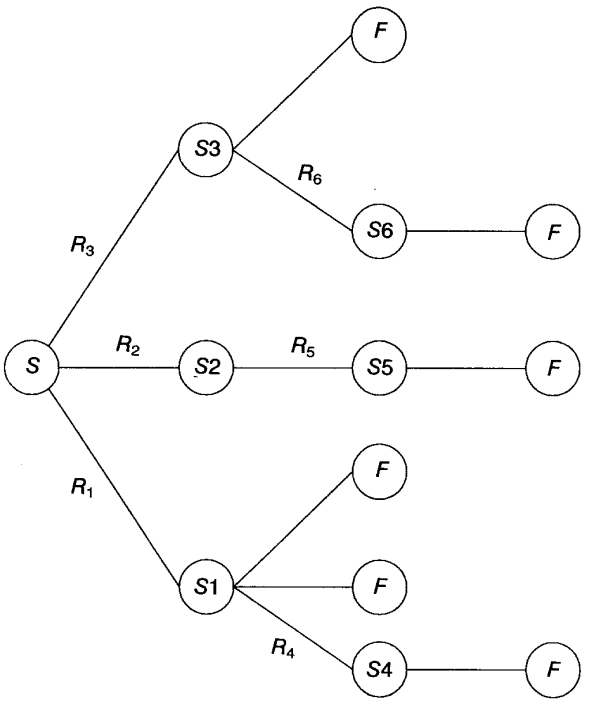

Fig. 2. State space search.

explicit search graphs that are actually constructed by the search program.

2.3. *Knowledge representation*

As a first step towards the development of the system, the knowledge representation scheme must be developed. This representation must be capable of storing the domain specific facts and rules. Furthermore, a representation of the product data has to be established. Once this is done, the control strategy can be implemented.

Knowledge used in an Expert System is of the following two types:

(1) Declarative knowledge – facts about objects, events and relations.

(2) Procedural knowledge – rules which interact with the facts to derive inferences.

An Object–Attribute–Value (O–A–V) structure is frame-like but differs in its handling of inheritance. In this system, facts are stored in a triplet, where attributes are characteristics of the object taking on a specific value. With O–A–V triplets, they work similar to slots. O–A–V triplets can be used to represent both static and dynamic aspects of the knowledge base.

The knowledge of this system is represented with production rules in the form

IF conditions **THEN** actions

The condition part (premise) has to be true for the action (conclusion) to be taken.

2.4. *Inference engine*

An inference mechansim of an expert system controls the sequence and application of the rules. The following are some of the important issues to be considered (Rich, 1986) in relation to the design of the inference structure:

(1) search direction;
(2) topology of search;
(3) rule matching;
(4) conflict resolution; and
(5) search strategy

2.4.1. *Search direction*

The objective of a search procedure is to discover a path through a problem space from an initial configuration to a goal state. There are two directions in which such a search can proceed:

(1) Forward chaining starts with the data and applies all the rules to the given data, until a solution is found or all the rules have been applied without any result.

(2) Backward chaining tries to prove a goal by setting up subgoals and tries to establish all the facts needed to reach that goal.

As redesign is based on analysis and applying rules to the available data to establish other facts, it is a data-driven problem. Since we have to provide suggestions based on certain facts, it is a forward chaining system. But, to provide these suggestions as our goal, we would have to prove other facts or set up sub-goals. Thus we use a combination of both forward and backward chaining to run through our rules and achieve our goal. This will become clear, once we talk about the actual implementation of the inference structure.

2.4.2. *Topology of search*

A simple way to implement any search strategy is as a tree traversal. Implementing such a tree traversal procedure is often simple and requires very little bookkeeping. However, this process often results in the same node being generated as part of several paths and so being processed more than once. Therefore, instead of traversing a search tree, traversal of a directed graph was employed. This graph differs from a tree in that several paths may come together at a node ('OR' graphs), which reduces the amount of effort that is spent exploring essentially the same path several times.

2.4.3. *Rule matching*

This stage of development is concerned with the scheme to extract rules for firing from the entire collection of produc-

tion rules. Sometimes, the problem of selecting applicable rules is greater than the simple problem of finding a way to ignore the bulk of the rules and go immediately to the appropriate ones. Most of the problem of rule matching is reduced by the way PROLOG uses the concept of *unification*, to match appropriate rules. Unification is a pattern-matching process, by which PROLOG tries to match a term against the facts or the antecedents of other rules in an effort to prove a goal.

We also use the concept of pointers, that link different sets of rules. There is an inherent hierarchy built into the building of the rule graph which makes different sets of rules candidates to be applied at different stages of the system.

2.4.4. *Conflict resolution*

Typically, the result of the matching process would be a list of rules whose left sides match the current state. It is the job of the search method to decide upon the order of application of the rules. This phase of the matching process, called *conflict resolution*, is very difficult to establish. Thus, it is useful to incorporate some of that decision making into the matching process itself.

The rule graph is built in such a way that we get over this problem to a certain extent, by having matching premises down the graph and conflicting premises across the graph. This would mean that we would not come across a situation where the search procedure would be confronted with multiple paths at a particular state. There would be one and only one candidate path or solution. If the system is unable to solve the problem, it would require input from the user to resolve the conflict.

2.4.5. *Search strategy*

Well-designed heuristic functions can play an important part in guiding a search process efficiently towards a solution. The more accurately the heuristic function estimates the true merits of each node in the search tree or graph, the more directed would be the solution process.

The way our system is built warrants the use of depth-first algorithm for an effective search mechanism. This concept would become clear once we discuss the actual implementation of the system.

2.5. *Evaluation of design improvement*

The final stage of development focuses on the approach in which design improvement can be evaluated. Since cost information would be difficult to acquire, a close approximation would be to provide an estimate of the savings in the average assembly cycle time to produce a complete product or assembly.

In the assembly analysis technique developed by Boothroyd and Dewhurst (1983), information is presented in the form of classification and data charts for robot assembly systems. For our study, we consider the single-station one-arm assembly system. In these charts, part placement, insertion or other required assembly operations are classified according to difficulty. For each classification and depending on the difficulty of the operation, relative time factors are given which can be used to estimate assembly times.

Typically, we would be able to infer two values, TP, the relative effective basic operation time, and TG, the relative time penalty for gripper or tool change. In their work, the basis for time estimates is the average time taken by the robot to move approximately 0.5 m, grasp the part, return and insert the part, where the motion is simple and no insertion problem exists. For a typical current generation robot, this process might take 3 s. However, if faster robots are available, the user can adopt a more suitable basic insertion time and all of the time estimates would change in direct proportion.

Now, we use the following equation to calculate the assembly cycle time for the different parts:

$$TA = TB(RP(TP + TR) + TG)$$

where:

TA = total operation time.
TB = basic operation time.
RP = number of times the above operations are repeated.
TP = relative effective basic operation time.
TR = relative time penalty for final orientation of the part by the robot.
TG = relative time penalty for a tool change.

The assembly cycle time is calculated for each and every part in the product, before and after redesign. Along with the cycle times, the design efficiency of the product is also calculated using the following equations:

$$EM = 3NM/TM$$

where:

EM = design efficiency.
NM = theoretical minimum number of parts.
TM = total assembly cycletime for the product.

This equation assumes an 'ideal' time of 3 seconds for assembly of any part that is easy to orient and align.

3. Implementation

In the development of the proposed system, the main focus of interest is the knowledge representation scheme and the inference mechanism, rather than building the knowledge base. Given the scheme for representing the domain specific knowledge and the mechanism for manipulating this knowledge, the system is limited only by the scope of the knowledge base, which could be expanded at a later

stage. Once this is completed, the user interface, explanation mechanism and knowledge acquisition module can be incorporated.

3.1. *Knowledge representation*

The reasoning mechanism or control strategy is influenced by the representation of knowledge and data. As mentioned earlier, knowledge can be divided into declarative (facts) and procedural (rules) knowledge.

3.1.1. *Static knowledge*

Declarative knowledge in the redesign domain is modeled as O–A–V triplets. These describe knowledge that do not change during a consultation session and are obtained initially. The following are some of the facts considered at the moment:

Every robot's performance is bound by certain characteristics such as configuration, number of axes, accuracy, payload capacity and gripper capabilities. The characteristics of the robot reflect on the types of parts that can be assembled by it. Thus, facts about the robot are implemented as O–A–V triplets as shown below:

```
robot(puma,configuration(articulate)).
robot(puma,envelope(12,15,320,275,300)).
robot(puma,gripper(two_finger,air)).
robot(puma,payload(30)).
robot(puma,accuracy(0.03)).
      .
      .
```

The configuration of an assembly robot is a spatial constraint involving basic mechanical configuration, arm sizes, work envelope, gripper type, maximum payload, accuracy, etc. The spatial constraint is inferred from the O–A–V triplet in the database.

Knowledge about the product and the parts that go to make up the product are also stored in the form of O–A–V triplets. Each part has several attributes and each attribute has an associated value. The name of a particular part is the *object* in this triplet. The following are some of the attributes being considered: part name, number of same parts, KK3 code, weight, mating parts, type of relation and accuracy, degree of symmetry and direction of insertion.

Knowledge about a particular part would be represented as shown below:

```
part(top_clamp,number(1)).
part(top_clamp,wt(2.2)).
part(top_clamp,mating_parts([
                (bolt,fit,0.03),
                (bracket,fit,0.03),
                . . .])).
      .
      .
```

Besides the above, as mentioned earlier, a CAD file describing the features of rotational parts is also used to describe a part. A preprocessor modifies the output of that file into a form suitable for our needs and stores the necessary information about the parts as follows (see Wang, 1986):

```
surface(shaft,1,ext_face,4.1,4.5,4.1,5.2,_,_,_).
surface(shaft,2,ext_cylinder,4.1,5.2,6.7,5.2,_,_,_).
      .
      .
```

3.1.2. *Procedural rules*

Procedural rules in this system are typically ease of assembly rules for design. The syntax of the rules is shown in Fig. 3. Using the *modus ponens*, the premise will imply the result, only if every condition in the premise is satisfied.

The premise may consist of different conditions which are connected by logical conjunctions or disjunctions. A condition is formed by the presence or absence of an attribute of an object. The conditions are in the form of an object, attribute and a value where the object has an attribute which can take on a particular value. The action part consists of a list of conclusions or actions, which will be executed only if all the conditions in the premise are true.

The rules are classified as to their field of application as follows:

(1) rules for feature interpretation.
(2) rules for checking parts with respect to robot capabilities.
(3) rules for determining assembly cycle times.
(4) rules for feeding, orienting, manipulating and inserting.

As mentioned early on in this paper, it is necessary to get information about a part such as taper, chamfer, internal diameter change, etc. to apply the redesign rules. Given the knowledge about the part in the form of all the surfaces making up the part, it is our task to interpret them to suit our requirements.

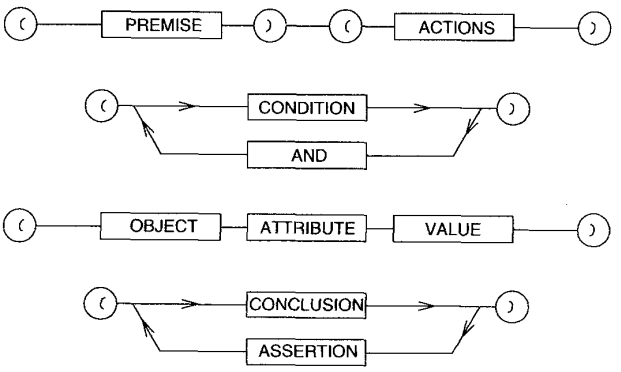

Fig. 3. Rule syntax.

A typical rule for identifying the presence of a chamfer might be stated as follows:

```
(Part,chamfered) :-
  (Part,surface_1) = ext_face,
  (Part,surface_2) = ext_taper,
  (surface_2,Y_end_pt) > (surface_2,Y_start_pt),
  (surface_2,X_end_pt) − (surface_2,X_start_pt) < 0.15.
```

A typical rule for redesign suggestion after a part is checked with respect to robot capabilities is shown below:

```
rule(wt_payload,more,
  [suggest("Breakup the part into multiple components, or
            consider change of material, or
            use another robot with larger payload.")]).
```

Assembly cycle times are needed for the sake of evaluating the proposed design. An example of such a rule is as follows:

```
rule_n :-
        attrib(Part,added_secured(Y)),
        attrib(Part,resistance(N)),
        attrib(Part,std_gripper(Y)),
        attrib(Part,holding_down(N)),
        attrib(Part,self_aligning(Y)),
        assert(Part,basic_time(1)),
        assert(Part,gripper_time(0)).
```

The above rule when interpreted, means that the basic time factor is 1 and gripper time factor is 0 for the above combination of conditions. Thus, for different combination of conditions, different times are obtained.

The assembly cycle times obtained from the *Design for Assembly* is stored in the form of a *binary* tree as shown in Fig. 4 (Boothroyd and Dewhurst, 1983). Every node represents either some attribute of the part, such as direction of insertion, resistance to insertion or capabilities of the robot such as standard gripper or not. Each node has two arcs leading to their successor nodes. Depending on the value a particular attribute takes, one or the other arc is taken to traverse down to the next node. This process is continued until we encounter a node that does not have any successor, where we get our cycle time for the particular part.

Rules for ease of orienting and feeding are usually based on the main part shape (α and β symmetry of a part) and presence or absence of certain features or attributes in a part, such as chamfer, taper, stepped hole, etc. These rules are again represented as productions but are implemented in the form of a directed graph.

```
rule_c2 :-
        part(spring,num(1)),
        part(spring,end(open)),
        suggest("provide close ended spring").
rule_c2 :-
        part(spring,num(1)),
        part(spring,dia_pitch(less)),
        suggest("increase diameter or decrease pitch").
```

If the inference engine has to go through all the rules in the system every time, there is unnecessary instantation of the same predicates or conditions. To get over this problem, these rules were implemented in the form of a graph as depicted in Fig. 5. Here again, each node represents a condition or an attribute of the part and the arcs represent the different values that it can take. Each arc is also associated with a suggestion list and an assertion list. The suggestion list is in the form of suggestion for redesign.

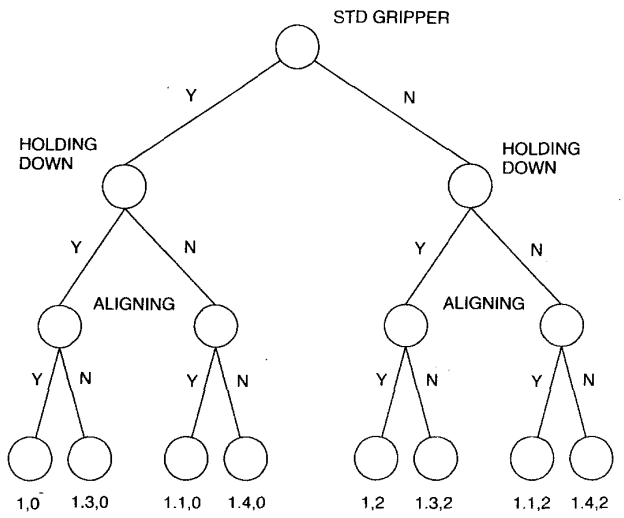

Fig. 4. Binary tree.

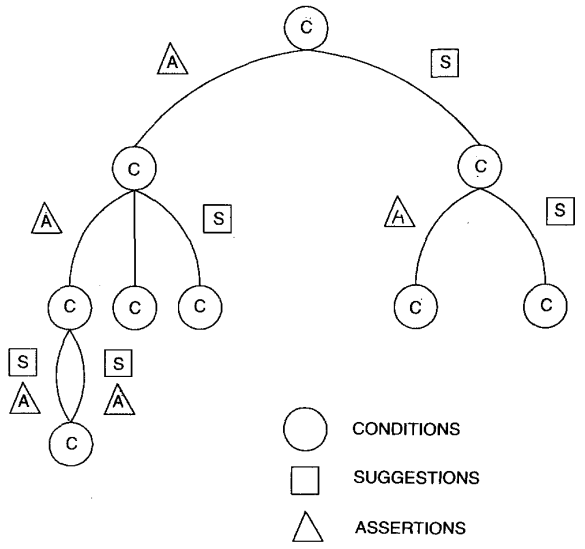

Fig. 5. Rule graph.

The assertions are basically facts about the parts that were inferred during the consultation and goes into the dynamic database for later use by the system. The assertions are necessary to help us deduce the cycle times for the redesigned parts. The following shows how a typical set of rules are implemented in the system:

```
rule(fastener_type,screw,[assertx(both,"std gripper",no),
                          assertx(both,"snap fit",no)],point).
rule(point,pilot,[assertx(both,"self aligning",yes)],nil).
rule(point,flat,[suggest("Use cone, pilot or oval point screw."),
                 assertx(before,"self aligning",no),
                 asserx(after,"self aligning",yes)],nil).
```

The graph is so structured that conditions across the graph are conflicting and conditions down the graph are matching. As shown above, each node has a pointer to the next node or condition that has to be checked. The advantage of implementing the rules in this graph structure is that, once we visit the nodes down the graph, we would have provided all the related suggestions without repeated instantiation or checking of the same premises over and again.

3.2. *Inference engine*

Once the knowledge is implemented in the system, the actual application of production rules (rule firing) is controlled by the inference engine. The algorithm used by the engine to evaluate the different rules is as shown below:

```
begin{infer}
    current node <== root node,
    while current node ≠ terminating node,
    do
        determine value of attribute associated with current node,
        select arc matching that value,
        provide suggestions, if any
        make assertions, if any,
        traverse arc to next node,
        current node <== next node
    end do
end while
end{infer}
```

The above algorithm is depicted in the form of a flow diagram as illustrated in Fig. 6.

At the beginning, the inference structure starts evaluating the first condition of the rule. It checks whether all necessary information for testing the premise is available. If not, the search module is invoked to gather the necessary information. Once the information is gathered, a check is made to see whether there are any matching premises. If there is any matching premise, the associated suggestions and assertions are made, if any. These assertions are used by the system in inferring new knowledge about a particular part that was not evident at the beginning of the consultation.

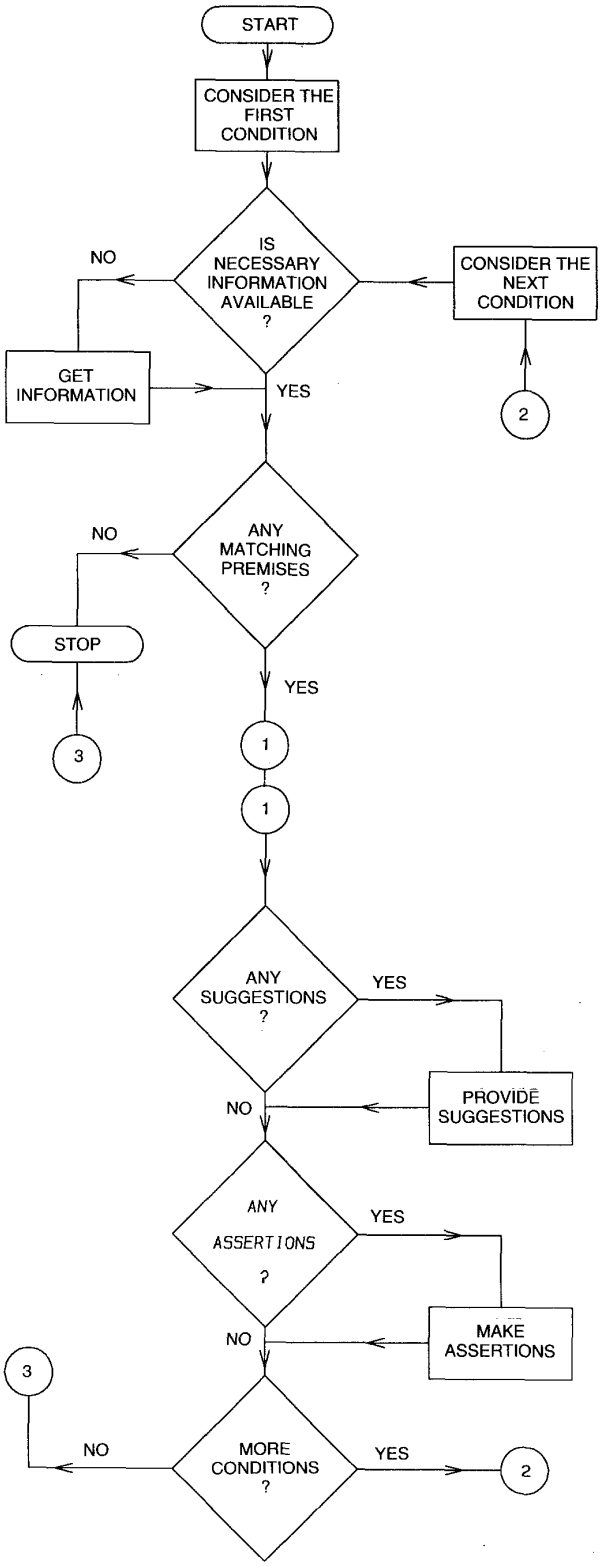

Fig. 6. Flow chart of the inference engine.

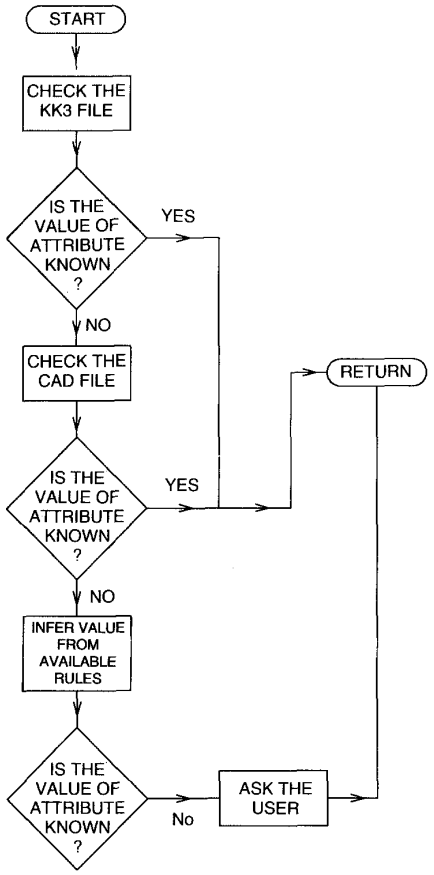

Fig. 7. Flow chart of the search mechanism.

Next, it will be examined to see whether there are more conditions to be evaluated. If so, the next condition will be considered for testing and the procedure is called recursively. If there are no more conditions to be tested, the system gathers all the suggestions and exits.

Once the search module is invoked by the inference mechanism as shown in Fig. 7, the module first infers the main part shape, α and β symmetry, weight, accuracy, material, etc. based on KK3 code and postprocessed CAD data. It then checks whether the value of part attributes (taper, slot, etc.) could be inferred from the KK3 code of the particular part. If so, it would check the KK3 code and return with the value of the attribute. If not, a check is made to see whether the value could be got from the CAD postprocessor file. Next the system checks to see whether any rules are present, that could possibly lead to the answer. Both design for assembly rules (α and β symmetry) and feature identification rules (attributes) are employed.

The concept of backtracking is used to deduce the value of the attributes from other rules. If the system is unsuccessful in deducing the value from the available rules, it comes back to the user for the value. Thus the user is not posed with unnecessary questions which could have been inferred by the system in the first place. At present the system does not handle uncertainty, but can be incorporated at a later stage.

At present, the knowledge acquisition module, explanation mechanism and the user interface modules are in a very nascent stage. But, since building expert systems is an incremental process, these modules can be built upon the existing framework of the system at a later stage.

3.3. *Example and discussion*

The redesign methodology for a product will be illustrated by the assembly of a ball bearing journal bracket for a ceiling fan, shown in Fig. 8. The bracket was originally designed to take two clamps between which the bearing was held.

Now, considering the present product, there might be scope for eliminating parts. One of the clamps appears to be redundant, because, the clamp and the bearing bracket need not be of different material, they need never move relative to each other and they need never be taken apart. Therefore, the top clamp and the bracket are combined, so that the operations of machining the clamp and screwing it on the bracket are eliminated. While one of the clamps is eliminated, the other clamp is necessary, since the clamp has to be taken apart to fit the bracket with the ball bearing which is standard and has to be of a different material than the bracket.

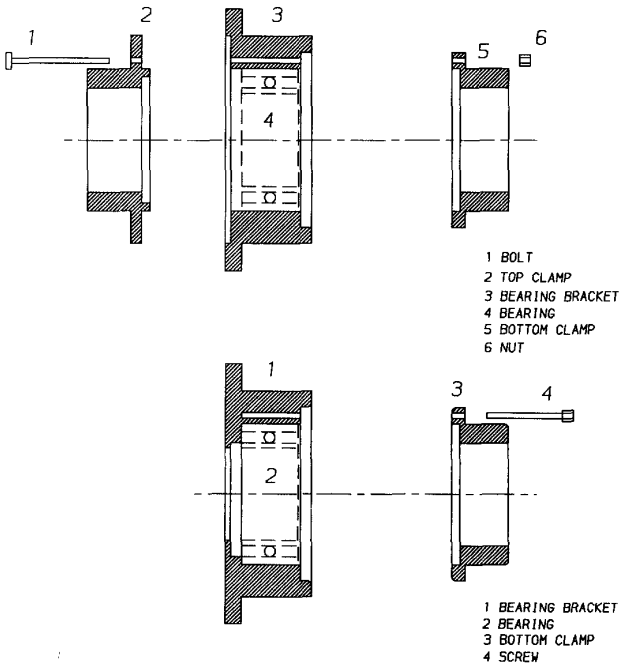

Fig. 8. Component design.

Initially, the above parts are held together by a bolt and nut assembly. Usually, parts whose only function is to fasten other parts are potentially redundant. The best situation would be totally to eliminate the nut and bolt assembly and provide snap-on or push-fit parts, but since there is a shaft which is going to have relative motion with respect to the above parts, the force exerted might be too high for a snap-fit part to handle.

Thus, the next best alternative would be to provide for a one-handed adjustment, by having screw connections instead of a bolt and nut assembly. Considering that screws were used to fasten the parts together, the next step would be to design the right screw point for ease of assembly. Screws should be provided with cone or oval points so that it is easy to locate and align.

Once the product has been examined for the necessity of each part, we get a theoretical minimum number of three parts, the bracket, bearing and clamp, with three screws to connect these parts together. Now these parts have to be analyzed to discover whether problems would arise during feeding, orienting, manipulating or inserting the parts.

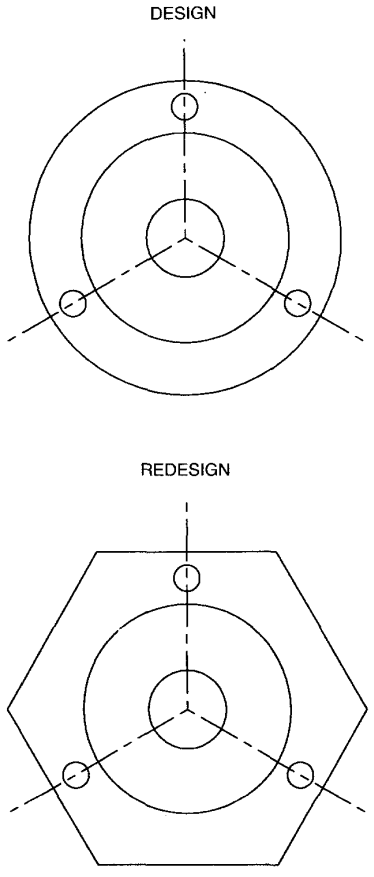

Fig. 9. Providing flats to aid in orientation.

One of the prime objectives was to provide for stacked or layered assembly and to reduce the number of directions of insertion. Originally, the parts have to be either inserted horizontally or the whole product has to flipped over, to connect them together with the bolt and nut. In the redesigned product, the parts can be inserted along the vertical direction, starting with the integrated bracket and then the rest of the parts can be stacked on in the vertical direction from the top.

Further examination shows that the clamp is easy to handle, but a little difficult to assemble. It seems likely that the orientation of the clamp will cause problems, since the clamp is asymmetrical about its plane of orientation. It would be best to make the clamp symmetrical, by providing a fourth screw hole, but that is going to increase the number of parts, which is undesirable. The next best alternative would be to aid orientation by providing tabs or flats as shown in Fig. 9. This would reduce the number of possible orientations of the part. Furthermore, it is best to eliminate sharp corners, by providing generous tapers and chamfers on the mating parts, which would aid in the insertion process.

The redesign analysis is shown in Table 2 and Table 3. Thus, the output from the program lists all the parts in the new design along with their individual assembly cycle times. The total number of parts along with the total assembly cycle time is also provided. The total estimated assembly cycle time for the new design is 31.5 seconds. As before, the theoretical minimum number of parts is six as compared to ten in the original design. This gives a design efficiency of 57 per cent and a saving in time over the original of 23.4 seconds.

4. Concluding remarks

Since assembly operations account for more than half the production cost for a particular product, there has been an increased interest in the field of assembly automation. The area of design for assembly deserves due attention, because of the impact it has, over the overall operating costs of a manufacturing plant.

Since the product is the key to efficiency of the assembly system, it is important that the design of the product is in sympathy with automated assembly practices and philosophies. To this effect, rules were developed to advise the designers on how to improve the 'assemblability' of a product, but are often overlooked. It was clear therefore, that in order for the designer to solve the assembly problem, the designer needs access to a higher level of automatic knowhow than that available in these rules.

Moreover, most knowledge in this domain existed as rules-of-thumb and as acquired experience. One solution to code this knowledge into the computer, was to use the

Table 2: Analysis before redesign

Part	No.	Cycle time(sec)
Top clamp	1	6.0
Bracket	1	6.0
Bottom clamp	1	6.0
Bearing	1	2.4
Bolt	3	19.5
Nut	3	15.0
Total assembly cycle time		54.9
Total number of parts		10
Design efficiency		0.33

Table 3: Analysis after redesign

Part	No.	Cycle time(sec)
Bracket	1	4.8
Bottom clamp	1	4.8
Bearing	1	2.4
Screw	3	19.5
Total assembly cycle time		31.5
Total number of parts		6
Design efficiency		0.57

concept of expert systems, an emerging technology that is making its impact felt in almost every facet of modern life.

In keeping up with the objectives of this research, this system was developed, and it shows that the principle of an expert system can be applied to the domain of redesign for ease of assembly. An attempt was made at eliciting geometric data about the parts, directly from a CAD post processor, in conjunction with the KK3 code. Although, at present, the above concept works only for rotational symmetric parts, an attempt was made to show how this system can be integrated directly with a CAD system.

The concept of O–A–V triplets were used to represent most of the domain specific knowledge. The whole expert system was implemented as a production system, which consists of rules or productions to represent relationships in terms of condition–action pairs. O–A–V triplets could be used to represent, both static and dynamic aspects of the knowledge base.

The main focal point of this study was to develop an effective control strategy or inference mechanism, to work with the facts and rules. The problem-solving process was structured as a search through a state space. A simple way to implement any search strategy is as a tree traversal. However, this process often results in the same node being node generated as part of several paths. Thus, a directed graph was used to implement the rules. Treating the search process as a graph search rather than as a tree search, reduces the amount of effort that is spent exploring essentially the same path several times.

One of the goals of this system was to produce a design with a minimum number of parts and to minimize the total assembly cycle time for the product. In the previous section, the redesign of a bearing bracket journal clamp was discussed in detail. In keeping up with the objectives of the system, the theoretical minimum number of parts was reduced from ten to six. All of the aforementioned design principles were incorporated in the redesign of the product. The total assembly cycle time of the product was also reduced considerably.

In conclusion, the redesigned product suggested by the system is good from a robotic assembly viewpoint. However, other factors such as manufacturability, cost, etc. should also be taken into account, in order to get a viable product design.

4.1. *Future directions*

Although this research has considered many aspects of the problem of integrating AI with the domain of redesign, there is still a whole range of avenues, yet to be explored. This section briefly reviews some of these ideas, in the hope that they may trigger future research in this area.

In this research, the redesign of a journal bearing clamp assembly was considered. The first step in improving the performance of the system is to enlarge its knowledge base, to encompass a wider variety of products and parts.

Considerable improvement can be made in the field of knowledge acquisition. Since technology is always changing, constant changes take place in the domain-specific knowledge and the knowledge base has to be either updated or modified. Thus, an efficient way of updating the stored knowledge base is called for.

Two other modules that require consideration are the explanation module and the user interface. Since a large amount of interaction takes place between the system and the user, the system has to be user friendly. The system should be able to explain its line of reasoning at any time. Since a lot of uncertainty is bound to be involved in the user responses, the system should be able to incorporate uncertainty in its reasoning.

One other way to improve the user interface is to provide graphical output in the form of the drawing of the redesigned product. This can be achieved by integrating the system with a CAD system, so that a common database could be accessed by both the systems. For this research, knowledge about parts was extracted from a CAD post processor, but only rotational parts were considered. A more comprehensive CAD post processor should be avail-

able so that other parts, like prismatic parts, could also be considered.

In relation to the above premise, the KK3 code was used to describe the parts. KK3 code is basically a machining code, which classifies parts according to their machining features. A comprehensive code, solely for the purpose of grouping parts with respect to their ease of assembly, is yet to be developed. Once this is done, much of the data input would be reduced to a great extent.

References

Andreasen, M. M., Kahler, S. and Lund, T. (1983) *Design for Assembly*, UK, IFS (Publications) Ltd.

Becker, T. (1986) 'Expert system for design for assembly', Unpublished Masters Thesis, Department of Industrial Engineering, University of Rhode Island, Kingston, Rhode Island.

Boothroyd, G. and Dewhurst, P. (1983) *Design for Assembly: A Designer's Handbook*. Department of Mechanical Engineering, University of Massachusetts, Amherst, Mass.

Boothroyd, G. and Redford, A. H. (1968) *Mechanical Assembly*. New York: McGraw-Hill.

Chang, T. C. and Wysk, R. A. (1985) *An Introduction to Automated Process Planning Systems*. Englewood Cliffs, New Jersey: Prentice Hall.

Dewhurst, P. and Boothroyd, G. (1983) Computer Aided Design for assembly. *Assembly Engineering*.

Field, R. P. and Russel, G. A. (1980) An Interactive Digital Computer Program to Encode and Analyse the Manual Assembly of Small Parts, *NSF Report APR 77 10197 7*.

Kumara, S. R. T. and Kashyap, R. L. (1985) 'Knowledge Representation in Expert System via Entity-Relationships and its application', *Proceedings 1985 IEEE International Conference on Systems, Man and Cybernetics*, Tucson, Arizona, November 12–15, 1985.

Rich, E. (1986) *Artificial Intelligence*, New York: McGraw-Hill.

Riley, F. J. (1982) The look of automatic assembly. *Manufacturing Engineering* **89**, 8.

Schraft, R. D. and Bassler, R. (1984) 'Possibilities to Realise Assembly Oriented Product Design', *Proceedings the 5th International Conference on Assembly Automation*, Paris, France, May 1984.

Swift, K. G. (1987) Design for automation in assembly, *Knowledge-Based Design for Manufacturing*, Englewood Cliffs, New Jersey: Prentice Hall.

Wang, H. P. (1986) 'Intelligent reasoning for process planning', Unpublished PhD Thesis, The Pennsylvania State University, University Park, PA.

Wang, H. P. and Chang, H. (1988) Automated classification and coding, *Advanced Manufacturing Technology* **2**, 25–38.

Wood, B. O. (1985) 'Design for Robotic Assembly', Unpublished Masters Thesis, Department of Industrial and Management Systems Engineering, Pennsylvania State University, University Park, PA.

Design for Manufacture, New Jersey: Prentice Hall.