Knowledge Representation System for Assembly Using Robots

A. Jain and M. Donath
University of Minnesota
Minneapolis, MN 55455

1.0 Introduction

Assembly robots combine the benefits of speed and accuracy with the capability of adaptation to changes in the work environment. However, an impediment to the use of robots is the complexity of the man-machine interface. This interface can be improved by providing a means of using apriori knowledge and reasoning capabilities for controlling and monitoring the tasks performed by robots.

Robots ought to be able to perform complex assembly tasks with the help of only supervisory guidance from human operators. For such supervisory guidance, it is important to express the commands in terms of the effects desired, rather than in terms of the motion the robot must undertake in order to achieve these effects.

A suitable knowledge representation can facilitate the conversion of task level descriptions into explicit instructions to the robot. Such a system would use symbolic relationships describing the apriori information about the robot, its environment, and the tasks specified by the operator to generate the commands for the robot.

However, the knowledge representation system should provide a scheme for building assembly models in an environment that maintains the knowledge of the spatial and functional relationships among the engineering parts comprising an assembly. The system then prevents the user from violating these relationships unless specifically asked to do so. Thus, the knowledge representation system has a memory for specified constraints and prevents actions whose side effects cause the violation of these constraints.

The modeling method provides an organizational structure that maps the relationships between the components of an assembly into a set of constraints. These constraints are simplified using a number of rules to determine the available degrees of freedom for a component. When a new relationship is to be established, the required motion is calculated and the feasibility of this motion is determined by checking against the available degrees of freedom.

2.0 Task Level Programming

Task level programming is an attempt to make use of structured programming together with an information base that enables the use of apriori knowledge to interpret the commands issued by the programmer. Examples of task-oriented programming languages include AUTOPASS, RAPT and LAMA.

The processes are expressed in terms of the desired effects on the objects. Thus, a task is completed not when a manipulator has completed a series of motions, but when the objects have reached the desired configuration. The knowledge representation system converts the task level specifications into manipulator level commands.

AUTOPASS (1) accepts the steps of assembly as input from the user and with the help of a model of the world, converts these into a program that a manipulator can process. However, the system does not recognize changes in relationships occurring due to manipulator actions. The user is responsible for specifying physically realizable operations and also for informing the system of changes in the attachment relationships.

RAPT (2), developed at Edinburgh, has concentrated on specifying the spatial relationships and reasoning about them. The relationships between objects such as "B1 against B2" and "C1 against C2", etc., are converted into algebraic equations. These equations can be expressed in terms of operational parameters, such as joint angles of the manipulator or in terms of the degrees of freedom for the objects. The solution of these simultaneous non-linear equations is quite complicated and time consuming. However, some efforts at recognizing stereotypes and applying standard solutions have been successful. The emphasis here has been on representation of relationships; however, the geometrical representation is at present incomplete. Hence, path planning, collision detection, etc., cannot be implemented using this system alone.

LAMA (3) uses general program plans that are expanded by details of individual assemblies. Polyhedral representation for objects permits trajectory planning and collision detection. The constraints are expressed in terms of parameters that are unspecified elements of partially filled transformation matrices. The constraints are represented as constraint planes, which indicate the boundaries of permissible motion. Ranges of parameters are calculated using volume interactions and constraint plane interactions. The assumption is made that

cont. on p. 113 →

different constraints on an object are non-interfering. This assumption is not required in the system we propose, as the bodies always move within the specified constraints and inconsistent constraints cannot be established in a constraint enforcing environment.

LM (4) and Feature Descriptor (5) represent other examples of task level programming.

Fahlman (6) proposed a task planner for robot construction tasks in a blocks' world domain. The information contained in the object's motion was not used. The feasibility of the final state is determined from a stability viewpoint, but the motion from the initial to the final state is not considered. A task planner built in a knowledge base that considers the feasibility of object motion can be integrated with a trajectory planning system.

In the proposed knowledge representation system, a constraint enforcing environment is provided at the level of the database. This approach permits supporting a task planner as well as interactive sessions with the programmer. This approach varies from the traditional approach by transferring some supervisory capability to the computer. The feasibility of motion in a constraint enforcing environment is determined by mapping relationships into a constraint set, simplifying the constraint, and interpreting the resulting constraints.

3.0 Geometrical Model

Homogeneous transformations are used to express the position and orientation information for objects. The position and orientation with respect to the world reference frame is stored with each object. Hence, this information is updated only when the object moves. The primitives, *e.g.*, blocks and cylinders, are defined in terms of the centroid position and faces. Loci definitions are used as they are sufficient for the reasoning involved in constraint simplification as discussed later.

An assembly is defined recursively as an object consisting of other objects. By imposing the restriction that these primitives cannot move relative to one another, we can create rigid assemblies. We can represent mechanisms by permitting specified relative degrees of freedom for the primitives that combine to form the assembly.

4.0 Relationships

The objects can be related to one another in a variety of ways with regard to relative motion. These relationships are shown in Figure 1. The contact relationships involve both concave and convex surfaces. A contact relationship between two convex surfaces is called "against". The against relationship can involve a surface, edge, or a point contact between two bodies. A contact relationship between a concave and a convex surface is called a "fit".

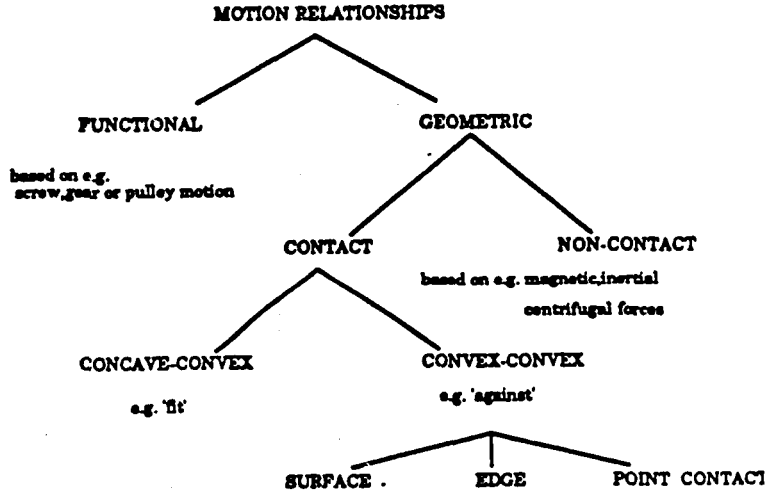


Figure 1: Motion Relationships

4.1 The "Against" Relationship

We will first define "against" in a restrictive sense and then generalize the definition. "Against" is a relationship established between two plane surfaces, such that translation is permitted in the common plane and rotation is permitted about an axis normal to this common plane. An example of against relationship is shown in Figure 2. This "against" relationship exists between face F1 of block B1 and face F2 of Block B2.

This definition of "against" can then be extended to curved convex surfaces, where the contact area reduces from the plane to a line. The object with the curved surface retains any rotation capabilities that it had about

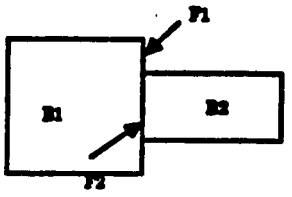


Figure 2: B1 Against B2 (F1, F2)

the body axis normal to the curved cross-section, prior to establishing the "against" relationship. Similar definition applies for spherical surfaces where the contact reduces to a point.

4.2 The "Fit" Relationship

"Fit" is a relationship established between a concave surface such as a hole and a convex surface such as a shaft. "Insert" is the driver function to command the establishment of a "fit" relationship. The hole and shaft can be of various cross-sections. The pre-condition for establishment of a "fit" relationship is that the profiles of the convex and the concave surfaces must match.

In a "fit" relationship, the body containing the hole and the body containing the shaft are restrained to rotate about the hole-shaft axis if the profiles are circular. A translation along this axis is also permitted.

Many special cases of "fit" can then be defined. A restricted "fit" relation might permit only rotational freedom. A rectangular key, on the other hand, is permitted translation but no rotation.

Representation of other relationships such as threaded joints, etc., can be accomplished by using these basic relationships. Permissable translation and rotation would no longer be independent of each other, but would be related by parameters such as the thread pitch.

The contact relationships of "against" and "fit" and other functional relationships in the motion domain map into a set of motion constraints that represent the degrees of freedom for the objects. These constraints are represented in our system in terms of new primitives which are necessary for reasoning about motion. These new constraint relationships represent a general set into which all the relationships that pertain to motion can be mapped. An example of this mapping is shown in Figure 3.

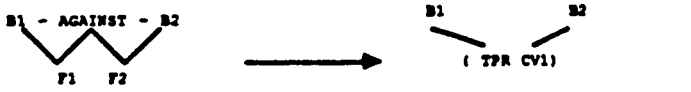


Figure 3: Mapping of Relationships to Constraints

5.0 Drivers

The driver functions that command the motion execution are of two types. The first type is the "move" function. The "move" function is used when some explicit motion is to be implemented by specifying the amount of motion or by specifying the final destination. The second type of drive functions are those which command the establishment of relationships. These functions determine the motion required to establish the relationship, find out if the motion is possible, and, if so, then send out the necessary instructions to the robot to implement the motion. For the two contact relationships, "against" and "fit", the driver functions are respectively called "establish-against" and "insert".

Determination of motion feasibility involves a number of steps. This is shown in Figure 4. The relationships of the body with other bodies, represented as constraints, are simplified to determine the translational and rotational degrees of freedom. If the entire desired motion is not possible with the available degrees of freedom, then an attempt is made to find one or more neighboring bodies such that this set of bodies can together accomplish the remaining motion. This strategy is explained in Section 8.

6.0 Representation of Motion Constraints

We need a representation that is sufficient for the reasoning involved in object manipulation, and is also simple and efficient to manipulate. We have departed from the conventional manner (Popplestone, Taylor, Mazer, etc.) of converting all the constraints into parametric equations and solving them over all the objects under consideration.

Let us define a few symbols first:

T: represents a Translation degree of freedom
R: represents a Rotation degree of freedom
L: stands for a Line. This is used as a mnemonic for the axis of rotation or a line of translation.
P: stands for a Plane.

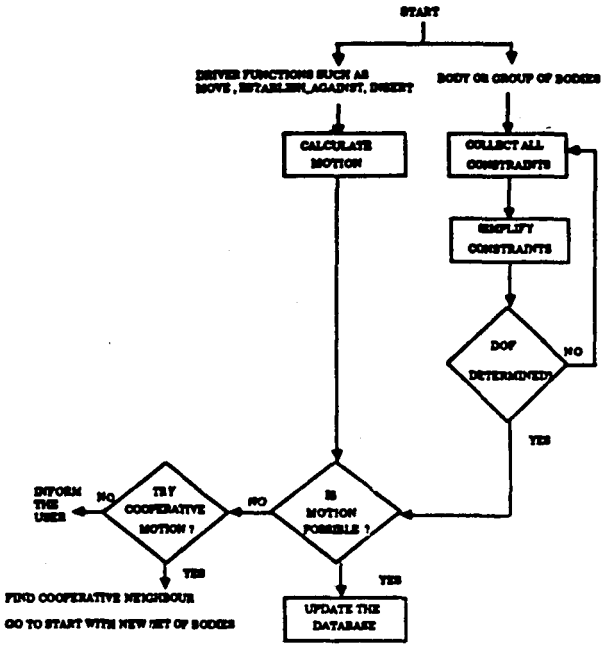


Figure 4: Feasibility of Motion

These mnemonics are combined in the following ways to give the constraints that are established by "establish-against" and "insert" operations:

TPR: This code indicates that Translation is permitted in the Plane and Rotation is permitted about a normal to the Plane. The normal to the Plane must be specified to complete the description of the constraint.

TLR: This code indicates that Translation is permitted along a Line and Rotation is permitted about that Line.

TL: This code indicates that Translation is permitted along a Line and no rotation is permitted.

RL: This codes indicates that Rotation is permitted about the specified axis.

The direction vector that is required with all of these codes is called the constraint vector (CV). Since the magnitude represented by this vector is insignificant, it is stored as a normalized vector. This direction is interpreted differently depending on the presence of P or L in the code.

Besides the above constraint primitives, we must define a few additional constraint classes in order to implement a working set. These include three other constraints--"Stationary," "Attached," and "Fixed"--and the unconstrained condition, "Free". These are clarified as follows:

The "Stationary" constraint indicates that the object has a particular position and orientation, and these cannot change.

Object A is said to be "Attached" to Object B when A and B always move together. In other words, A is rigidly connected to B. Every object must store a list of all other objects that are "Attached" to it. When the move command is to be executed for an object, all the "Attached" objects should move together.

As opposed to the "Stationary" and "Attached" constraints which are specified by the user, "Fixed" is a constraint derived from a set of user specified constraints. "Fixed" represents the condition when a combination of constraints applied to an object results in no freedom of movement for the object when considered as a single entity. However, the implication is that the object could move if it were to move together with one of the constrained objects. This is clarified by the example shown in Figure 5.

Consider the case when the following constraints are established:

B1 AGAINST B2 (F2 F1)

B2 AGAINST B3 (F3 F4)

The features involved are indicated within parentheses.

Under these two constraints, B2 can translate along a line in and out of the paper. Now, suppose we establish a third relation B2 AGAINST B4 (F5 F6) in which B4 is behind B2. The result is that B2 is now "fixed"

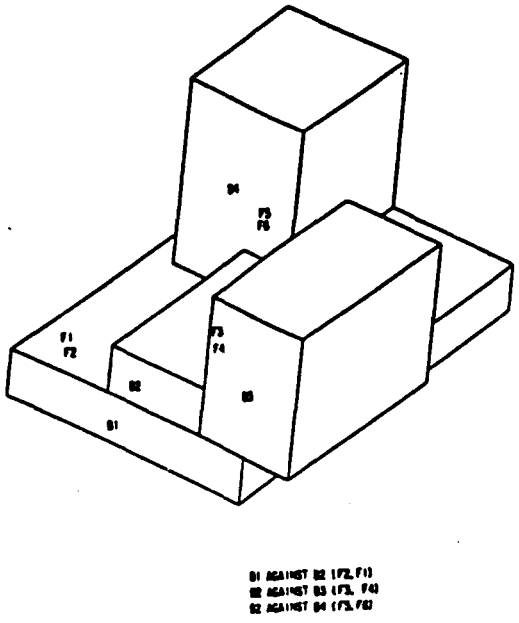


Figure 5: B2 "Fixed" By Constraints

and cannot move by itself. However, any of the combinations (B2,B4), (B2,B3), and (B2,B1) can move along different directions.

The constraints only describe the restrictions that the environment has on the object's motion. The driver functions, the "move" command, and the commands to establish relations use these constraints to determine whether the body can be moved or some other body needs to move along with the body prior to transmitting the command to the robot. By using these constraints, we can build assemblies in a constraint enforcing environment where the user or the task planner is prevented from violating any previously specified constraints, unless the system is specifically asked to do so.

In order to determine the available degrees of freedom, we will first show how to simplify the motion constraints in the next section. Following that, we will develop a method of determining the available degrees of freedom from the specified constraints.

7.0 Simplification of Constraints

Some rules for simplifying constraints are presented here. These apply to the case of rectangular blocks and cylinders. The extension to other bodies involving curved surfaces and to more general profiles is possible.

The rules may not represent a complete set, but are sufficient to demonstrate the feasibility of the approach. Thie rules which follow are for a prototypical object B1 which is being commanded to move by some driver function (→ signifies is mapped into):

1. IF (TPR CV1) AND (TPR CV2) AND CV1 X CV2 ≠ 0

   THEN (TL CV) AND CV = CV1 X CV1

Consider the motion for B1 in the following examples (Figure 6).

B2 AGAINST B1 (F2, F1) → (TPR CV1)

B3 AGAINST B4 (F4, F3) → (TPR CV2)

Since CV1 and CV2 are orthogonal to each other and lie in the plane, B1 can translate along a line in and out of the plane.

2. IF (TPR CV1) AND (TPR CV1) AND CV1 X CV2 = 0

   THEN (TPR CV1) OR (TPR CV2)

Consider the motion for B1 in the following example (Figure 7a).

B2 AGAINST B1 (F2, F1) → (TPR CV1)

B3 AGAINST B1 (F4, F3) → (TPR CV2)

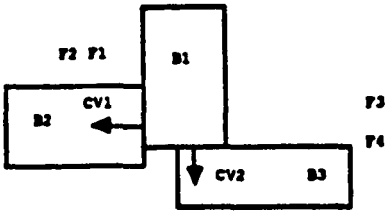


Figure 6: B2 Against B1 (F2, F1)
B3 Against B1 (F4, F3)

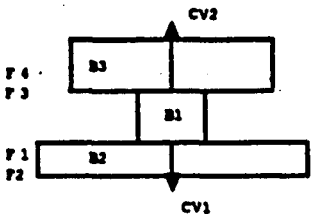


Figure 7a: B2 Against B1 (F2, F1)
B3 Against B1 (F4, F3)

B1 can still translate in the plane perpendicular to CV1 and rotate about CV1. Using CV1 or CV2 does not matter As shown in Figure 7b, CV1 and CV2 can also be pointing in the same direction.

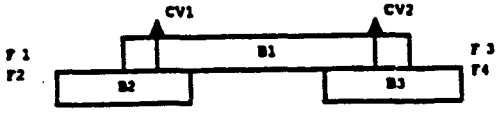


Figure 7b: B2 Against B1 (F2, F1)
B3 Against B1 (F4, F3)

3. IF (TLR CV1) AND (TLR CV2) AND CV1 X CV2 $\neq$ 0

   THEN FIX

Consider the motion for B1 in the following example for a "fit" relationship between a shaft and a hole (Figur 8).

B2 FITS B1 (S2, H1) $\rightarrow$ (TLR CV1)

B1 FITS B3 (S1, H3) $\rightarrow$ (TLR CV2)

B1 cannot move all by itself. However, B1 and B2 can move together as can B1 and B3.

4. IF (TLR CV1) AND (TLR CV2) AND CV1 X CV2 = 0

   THEN (TLR CV1) OR (TLR CV2)

Consider the motion for B1 in the following example (Figure 9).

B1 FITS B2 (S1, H2) $\rightarrow$ (TLR CV1)

B1 FITS B3 (S1, H3) $\rightarrow$ (TLR CV2)

Until now, we have been considering infinite lines and planes and, hence, no mention has been made regarding th depth of the hole and the length of the shaft. These parameters will be introduced later as factors necessar for determining the possibility of establishing relations required for object manipulation.

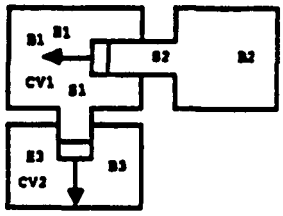


Figure 8: B2 Fits B1 (S2, H1)
B1 Fits B3 (S1, H3)

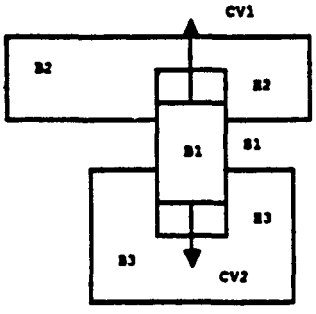


Figure 9: B1 Fits B2 (S1, H2)
B1 Fits B3 (S1, H3)

5. IF (TPR CV1) AND (TLR CV2) AND CV1 X CV2 = 0

THEN (RL CV1) OR (RL CV2)

Consider the motion for B1 in the following example (Figure 10).

B2 AGAINST B1 (F2, F1) → (TPR CV1)

B1 FITS B3 (S1, H2) → (TLR CV2)

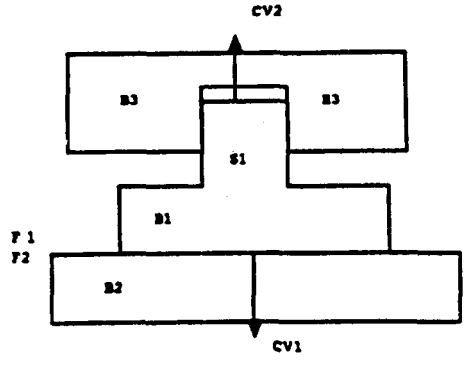


Figure 10: B2 Against B1 (F2, F1)
B1 Fits B3 (S1, H3)

B1 can rotate about CV1. B2 AGAINST B1 permits translation in a plane while B1 FITS B3 permits translation along the normal to the same plane. The net result is that B1 cannot translate at all.

6. IF (TPR CV1) AND (TLR CV2) AND CV1 X CV2 ≠ 0

THEN, IF CV1 - CV2 = 0, THEN (TL CV2), ELSE FIX

Consider the motion for B1 in the following example (Figures 11a and 11b).

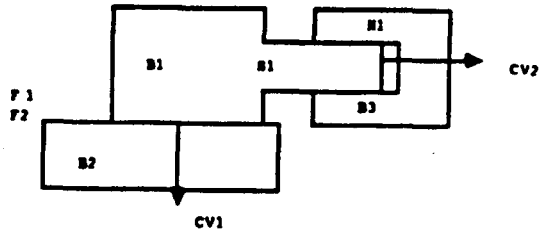


Figure IIa: B1 Against B2 (F1, F2)
B1 Fits B3 (S1, H1)

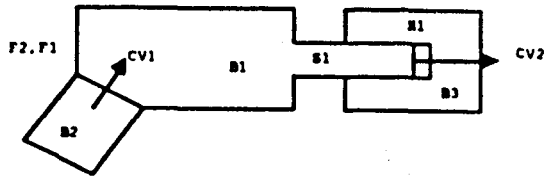


Figure IIb: B1 Against B2 (F1, F2)
B1 Fits B3 (S1, H1)

B1 AGAINST B2 (F1,F2) ⟶ (TPR CV1)

B1 FITS B3 (S1,H1) ⟶ (TLR CV2)

B1 can translate along CV2 as shown in Figure IIa for CV1 CV2 = 0. However, as shown in Figure IIb, when CV1 CV2 ≠ 0, B1 cannot move at all.

7. IF STATIONARY AND ANY ONE CONSTRAINT

THEN STATIONARY

If a body is stationary, it cannot move at all.

8. IF FIX AND ANY ONE CONSTRAINT EXCEPT STATIONARY

THEN FIX

If a body is constrained so that it cannot move by itself, then constraining it further will not change the FIX constraint.

9. IF FREE AND ANY ONE CONSTRAINT C

THEN C

Free refers to an unconstrained state and, hence, C is the only one constraint.

The relationships of a body with other bodies are stored symbolically using the constraint primitives discussed above. When motion feasibility is to be determined, the different relationships are first simplified using these rules. If a group of bodies moves together, then all their relationships with other objects are simplified in order to determine the feasibility of motion of the group as a whole.

The simplification is completed when we have all the possible translational and rotational degrees of freedom for the body or set of bodies. All the constraint vectors are converted into the world reference frame before simplification. This approach is preferred to updating the constraint vector every time the position of the object changes, since the number of updates required is reduced.

The next section discusses the implementation strategy for motion and for the establishment of relations.

8.0 Strategy for Implementing Motion and for Establishment of Relationships

Every primitive object has an arbitrarily selected reference point. The position of this point and the orientation of the body about this point as measured in the world reference frame is considered as the position of the object. The position and orientation of features are described relative to this reference point.

Let us first consider the motion of a primitive object. When the motion of this body to a particular

position and orientation is desired, the motion matrix is given by

$$\text{MOTION} = (\text{OLD-POSITION})^{-1}\ (\text{NEW-POSITION})$$

using the homogeneous transformation notation. This is represented by the diagram in Figure 12.

Figure 12: Calculation of Motion Matrix

8.1 Establishing Relations

Driver functions such as "establish-against" and "insert" are used to establish relations between objects. These commands are used to calculate the necessary motion matrix which is used to move the body in order to establish a relationship.

Consider the diagram shown in Figure 13. Let us define the terms used in this diagram:

POSB1:	position of object B1 in the world
OLD-POSB2:	position of object B2 before moving
NEW-POSB2:	position of object B2 after the relation has been established. This is undetermined at this stage.
MOTION:	The motion required of Object B2 to establish the relation. This is yet to be determined.
FEAT1, FEAT2:	position of features on bodies B1 and B2 that are involved in this relation.
REL-ESTABLISH:	a matrix that orients the constraint vectors to establish the relevant relation.

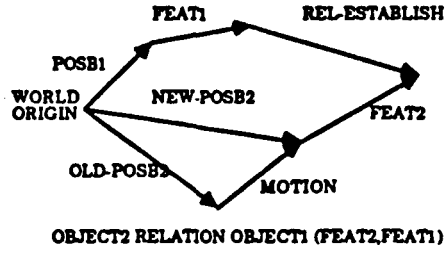


Figure 13: Establishing Relations

In the case of the "against" relationship, the REL-ESTABLISH matrix would define the rotation needed to bring the normal vectors associated with the two faces in line with each other. In the case of the "fit" relationship, it is the matrix that brings the two axial vectors in line with each other.

To determine REL-ESTABLISH, consider the example in Figure 14. Consider two unit normal vectors N1 and N2, associated with the faces F1 and F2 of two objects B1 and B2.

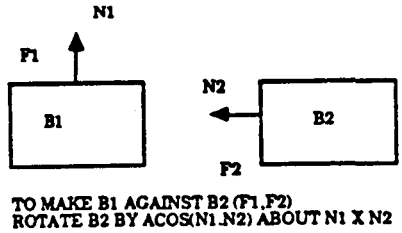


Figure 14: Definition of REL-ESTABLISH

N1 and N2 are two vectors to be aligned. A rotation of N1 by an angle $= \cos^{-1}(N1\ N2)$ about the direction N1 x N2 will bring them into line with each other. This rotation is the one defined by REL-ESTABLISH (Equation 1). An additional translation is required to bring the two faces together. NEW-POSB2 incorporates the combined rotation and translation of this new position for B2 (Equation 2). MOTION is calculated using this new position NEW-POSB2 (Equation 3).

$$\text{REL-ESTABLISH} = \begin{bmatrix} & & & 0 \\ \text{ROT} & \cos^{-1}(\text{N1}\ \text{N2}),\ \text{N1} \times \text{N2} & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

$$\left[\text{NEW-POS-B2}\right] = (\text{POSB1})\ (\text{FEAT1})\ (\text{REL-ESTABLISH})\ (\text{FEAT2})^{-1} \quad (2)$$

$$\left[\text{MOTION}\right] = (\text{OLD-POS-B2})^{-1}\ (\text{NEW-POS-B2}) \quad (3)$$

The following "loop equation" can be used at any time to determine if the relation has been established. If the equality is satisfied, then the relation has been established.

$$\left[\text{IDENTITY}\right] = (\text{POSB1})\ (\text{FEAT1})\ (\text{FEAT2})^{-1}\ (\text{POSB2})^{-1} \quad (4)$$

The next task is then to find out if this motion is feasible in the presence of the already established constraints.

However, there is one factor that still needs to be considered. The above-mentioned strategy for establishing relations will work only if object B1 does not move while object B2 is being moved. If object B1 moves in the process of achieving the required motion for object B2, then the relation is not yet established. This is explained by the following example (see Figure 15).

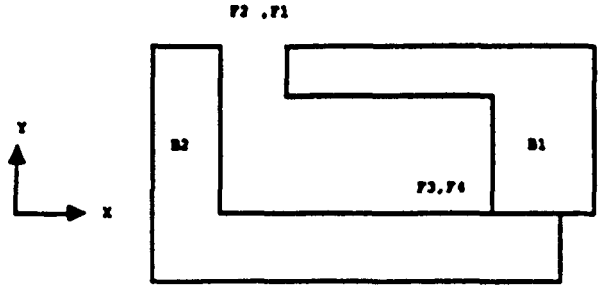


Figure 15: Example of Unachievable Relationship

The given constraint is B1 ATTACHED B2 (F3 F4). We try to establish the relation B2 AGAINST B1 (F1 F1).

The motion needed by B2 to establish this relation is calculated, and in this case, it is simply a translation in the x direction. When this motion is attempted, it is found to be feasible and, hence, is implemented. But since B1 and B2 are "attached", B1 will also move and, hence, the desired relationship will not be established. In fact, for this example, the relation can never be established since the two bodies are attached and, hence, move by the same amounts.

This problem is resolved by using an iterative procedure. Consider the diagram shown for the general case in Figure 16.

In the first attempt, motion M is the required movement of object B2 to establish the relation. However, in the process of planning motion M, motion M1 of object B1 also occurs. At the end of motion M, the "loop equation" (Equation 4) is used to check if the relation has been established. If the relation is established, the new motion matrix will indicate zero translation and rotation. If the relation is not established, then an iteration is performed in which NP2 is redefined as OP2, P1' is redefined as P1, the new rel-establish matrix is given by RE', and the motion matrix is recomputed. This is shown as M2 in Figure 16. By comparing the motion M with motion M2 (as shown later), one can find if any progress has been made towards establishing the relation. If no progress is made as in the example in Figure 15, then we conclude that the new relation cannot be established in the presence of the already established relation.

In the above iterative approach, it is necessary to determine if progress has been achieved in bringing the two objects closer to each other. We assume that the iteration is successful if the new motion prediction is "smaller" than the "initial" motion prediction. A mechanism to compare an initial motion prediction with a new motion prediction is described in more detail in reference (7).

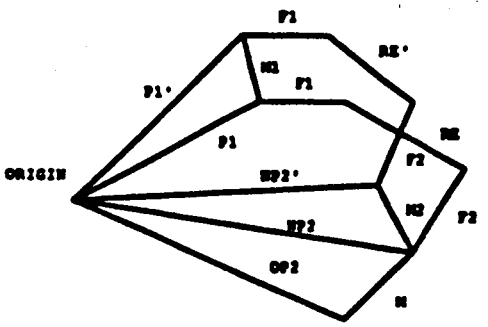


Figure 16:

P1 - old position of body 1
P1' - new position of body 1
M1 - motion of body 1
F1 - feature of body 1
F2 - feature of body 2
OP2 - old position of body 2
NP2 - new position of body 2
NP2' - new position desired for body 2
M - motion of body 2
M2 - motion needed for body 2
RE - old REL-ESTABLISH
RE' - new REL-ESTABLISH

8.2 Feasibility of Motion

In order to decide whether a particular body motion is feasible or not, we need to consider the constraints to which the body is subjected. We have shown how to simplify these different constraints in order to determine the possible degrees of freedom. In order to facilitate the process of determining motion feasibility, the translation and rotation are each handled separately.

If, after simplification has been performed, the constraint code includes TL, then a part of the translation needed can be accomplished by moving the body along the direction given by the constraint vector. If the constraint code includes TP, then a part of the motion is achieved by translating in the specified plane. If the entire desired translation is not possible, then one cannot achieve the entire motion by moving this object alone. One must then consider moving a neighboring object along with the body of interest. This strategy is described later.

The rotation part of a given motion matrix is converted into an equivalent angle about an axis located in the world reference frame. The constraint vector associated with the rotation degree of freedom is then compared with this equivalent axis. If the two coincide, then the rotation is possible; otherwise, the rotation is not possible by this body alone.

If the body is found to be constrained from moving when considered alone, then the following strategy for considering cooperative motion with neighboring objects is pursued. A search similar to a breadth first search is used. This is best explained by an example (see Figure 17).

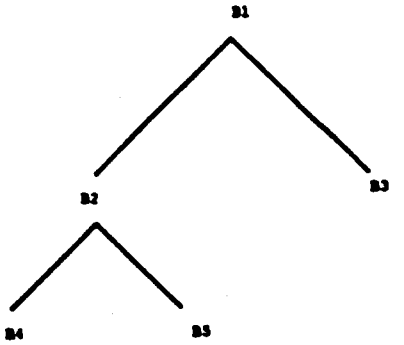


Figure 17: Cooperative Motion

Consider an attempt to move the object B1. Suppose the following relations already exist:

B1 is in contact with B2 and B3.

B2 is in contact with B4 and B5.

The first attempt is to find if B1 can be moved by the required amount all alone. If this is not possible, then an attempt is made to consider the motion of B1 and B2 together. In the constraint simplification that is performed, the constraints between B1 and B2 are not considered in this case as B1 and B2 do not have relative motion.

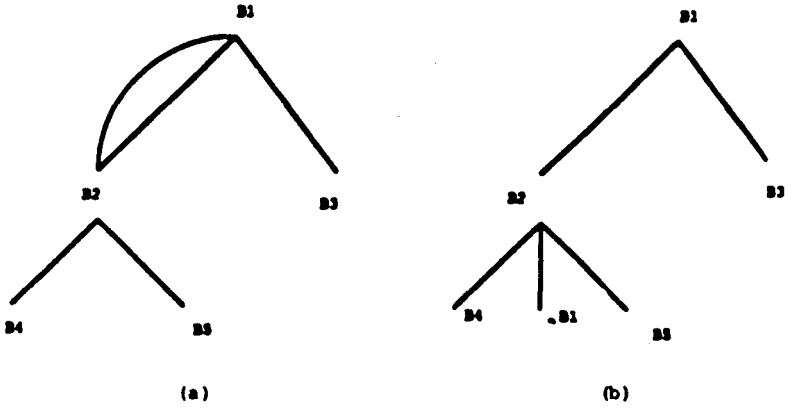
If B1 and B2 cannot accomplish the entire remaining motion together, then B1 and B3 are considered together. If the entire motion is still not implemented, then B1, B2, and B3 are considered together. The search order used is (B1), (B1,B2), (B1,B3), (B1,B2,B3), (B1,B2,B4), (B1,B2,B5), (B1,B2,B3,B4), (B1,B2,B3,B5), AND (B1,B2,B3,B4,B5).

The set (B1,B2,B3) is considered before (B1,B2,B4) since B2 and B3 are in direct contact with B1 which is the object of interest. The governing rule in this search is to always attempt to move a minimum number of bodies prior to attempting a larger set.

At each stage in the search, motion is carried out as far as possible. If there is still some motion required, then the search continues. As a result, the total desired motion for B1 might be achieved partly by B1 alone and partly by B1 and B3 together.

If a constraint "Stationary" is encountered in the search, the search along the path is immediately terminated. Note that all the attached bodies are always considered together in this motion feasibility paradigm.

The structure is described as a tree diagram. However, the generalized version can be a graph (see Figure 18a). This graph can be converted into a tree by repeating appropriate nodes (see Figure 18b).



(a) (b)

Figure 18: Conversion of Graph to a Tree

8.3 Motion of Assemblies

To determine the motion possibilities for an assembly, we have to treat them as an "attached" group of primitives. We store the position and orientation information for each primitive. When the assembly has to move, all the primitives must move by the same amount.

8.4 Range of Motion

In our discussion so far, we have considered the constraints as basically degrees of freedom. For instance, TPR indicates that translation is allowed in a plane and rotation is permitted about the normal to the plane. However, these motions are not infinite. These motions are limited by the finite dimensions of the bodies involved in the relationship.

9.0 Implementation

The knowledge representation system described here has been implemented using the FLAVOR system on the LMI LISP Machine. A typical data network is shown in Figure 19. In the constraints discussed above, we did not include the range of motion. Geometric algorithms to detect collision of objects and overlapping of faces, overlapping of two holes, etc., are implemented as tests to be conducted before sending any motion command to the robot. This scheme provides for expandability of the system to incorporate checks such as tolerance matching, surface finish matching, checking types of fit, calculating inertia properties, etc. Some examples can be found in reference (7).

The system described thus far is a constraint enforcing system. The breaking or making of relationships as a side effect of motion is not considered. This consideration of dynamic constraint propagation requires a constraint network on a global scale which was indeed designed and implemented (7), but is outside the scope of this paper.

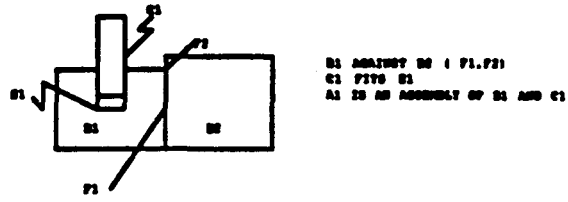
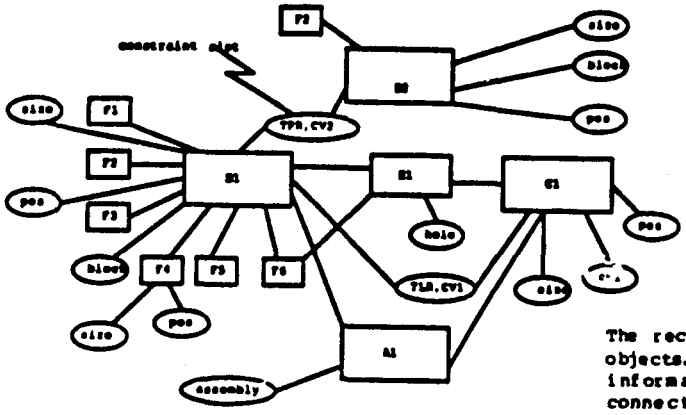


Figure 19a: Assembly Model



The rectangles indicate data objects. The ovals indicate information slots. Ovals connecting two data objects indicate information slots in both data objects.

Figure 19b: Data Network for Model Shown in Figure 19a

Note: This is not a complete network as some repetitive structures are not shown.

10.0 Conclusion

The applications of the knowledge representation system are manifold. The programming of robots using higher level constructs hides the robot specific programming details from the user by submerging them into the domain specific knowledge base. The constraint enforcing environment provides for an on-line debugging facility which operates at the conceptual task level rather than at the program's syntax level. Before sending a command to a robot, the command is simulated in the world model, and this provides for a real time error preventive capability. With the addition of dynamic constraint propagation, the knowledge representation system becomes capable of simulating simple assembly operations. This provides a basis for the development of a task planner.

11.0 Acknowledgment

This work was supported in part by National Science Foundation PYI Grant # DMC8351827.

12.0 References

1. Lieberman, L. I. and M. A. Wesley, "AUTOPASS: An Automatic Programming System for Computer Controlled Mechanical Assembly," IBM Journal of Research and Development, July 1977, pp. 321-333.
2. Popplestone, R. J. and A. P. Ambler, "A Language for Specifying Robot Manipulation," Robotic Technology (A. Pugh, ed.), Peter Perigrines Ltd., London, 1983, pp. 125-141.
3. Lozano-Perez, T., "Task Planning," in Robot Motion Planning and Control, edited by M. Brady, et al., MIT Press, 1982.
4. Mazer, E., "LM-GEO, Geometric Programming of Assembly Robots," Laboratoire IMAG, BP N68 38042, Saint Martin D'here, LEDEX.
5. Taskase, H. and N. Nakajima, "A Language for Describing Assembled Machines," International Symposium on Design and Synthesis, 1979.
6. Fahlman, S. E., "A Planning System For Robot Construction Tasks," Artificial Intelligence, Vol. 5, No. 6, 1976.
7. Jain, A. K., "Knowledge Representation For Automatic Assembly Using Robots," M.S. Thesis, Department of Mechanical Engineering, University of Minnesota, 1986.