9-1-1988

Task Planner for Simultaneous Fulfillment of Operational, Geometric and Uncertainty-Reduction Goals

S. A. Hutchinson
Purdue University

A. C. Kak
Purdue University

A Task Planner for Simultaneous Fulfillment of Operational, Geometric and Uncertainty-Reduction Goals

S. A. Hutchinson
A. C. Kak

TR-EE 88-46
September, 1988

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

A Task Planner for Simultaneous Fulfillment of Operational, Geometric and Uncertainty-Reduction Goals*

by

S. A. Hutchinson and A. C. Kak

Robot Vision Laboratory
School of Electrical Engineering
Purdue University
W. Lafayette, IN 47907

ABSTRACT

Our ultimate goal in robot planning is to develop a planner which can create complete assembly plans given as input a high level description of assembly goals, geometric models of the components of the assembly, and a description of the capabilities of the work cell (including the robot and the sensory system). In this paper, we introduce SPAR, a planning system which reasons about high level operational goals, geometric goals and uncertainty-reduction goals in order to create assembly plans which consist of manipulations as well as sensory operations when appropriate. Operational planning is done using a nonlinear, constraint posting planner. Geometric planning is accomplished by constraining the execution of operations in the plan so that geometric goals are satisfied, or, if the geometric configuration of the world prevents this, by introducing new operations into the plan with the appropriate constraints. When the uncertainty in the world description exceeds that specified by the uncertainty-reduction goals, SPAR introduces either sensing operations or manipulations to reduce that uncertainty to acceptable levels. If SPAR cannot find a way to sufficiently reduce uncertainties, it does not abandon the plan. Instead, it augments the plan with sensing operations to be used to verify the execution of the action, and, when possible, posts possible error recovery plans, although at this point, the verification operations and recovery plans are predefined.

* This work was supported by the National Science Foundation under Grant CDR 8803017 to the Engineering Research Center for Intelligent Manufacturing Systems.

TABLE OF CONTENTS

1. INTRODUCTION

Our ultimate goal in assembly planning is to provide the planner with a high level description of an assembly, geometric models of the components of that assembly, and a description of the capabilities of the robotic work cell (both the robots and sensors) and have the planner devise an assembly plan which can be executed to construct the assembly. Such a plan would include both manipulations and sensory operations. In order to achieve this goal, the planner must be able to reason at several levels. First, it must be able to determine the set of assembly operations which must be performed, and a set of precedence relations which determines a partial ordering on the execution of those operations. Second, the geometric specifications of those operations must be determined, and if these are not feasible, additional operations must be added to the plan to change the configuration of the world to make them feasible. Finally, the planner must take into account any uncertainties in its description of the world and in the robot's ability to execute actions.

Toward these goals, we have developed SPAR, a planner which considers operational, geometric and uncertainty-reduction goals. SPAR's approach to creating assembly plans is to first create a high level plan, containing actions like "pickup part-1", and then add constraints on the way these actions are executed, so that geometric goals are satisfied. It is also possible that additional high level actions will be added to the plan to satisfy geometric goals, for example, if a work piece must be repositioned so that an insertion operation can be performed. In order to satisfy uncertainty-reduction goals, SPAR examines the maximum uncertainty which might exist in its world description. If this uncertainty is too large to ensure successful execution of any action in the plan, sensing operations or manipulations are added to the plan in an attempt to reduce the uncertainty to acceptable levels. If this fails, rather than abandon the plan, SPAR adds sensing operations to verify the execution of the action, and when possible, adds precompiled recovery plans. By planning at these three levels, SPAR is able to start with a high level set of assembly goals and develop assembly plans which include geometric descriptions of the actions and sensing operations to reduce uncertainty and verify actions which might not succeed.

There are, of course, limitations to SPAR's planning abilities. First, SPAR only considers the "endpoints" of actions. Thus, if the plan calls for grasping an object and moving it to another place on the work table, SPAR will determine a set of constraints on the configuration used to grasp the object, and on the configuration used to place it on the table, but will not plan the motions required to move the manipulator from the first position to the second. Second, we have not incorporated any fine motion planning into our current planner. As a result of this, in some situations where compliant motion plans could be used to robustly perform an assembly task, SPAR will pessimistically declare that uncertainties are too great to guarantee successful assembly, and that error detection sensing should be used at execution time. Currently, there is research being done on in our lab on compliant motion planning [17], and at some future point that work could be integrated with this planner.

Another limitation of planning by SPAR is that the planner must know a priori about the locations and orientations of all the objects that participate in the assembly. This evidently implies that SPAR has available to it a sensor suite capable of generating information on the world around the robot. At this time, SPAR makes no checks on whether or not all the objects required for assembly are available in the world knowledge. However, we believe that augmentation of SPAR with such capability is simple; the hard part is the development of the required sensor technology. In a sense then SPAR could be considered to be ahead of its time, at least from the standpoint of its immediate utility in robotic assembly. Despite this shortcoming, the reader should note the fundamental contributions made by SPAR. For example, SPAR has shown how a reasoning architecture can be developed that combines planning at the symbolic level -- an area much pursued by many researchers before us -- with planning at the geometric level taking robot kinematics into account, while simultaneously reasoning about sensor invocations to reduce any hypothesized uncertainties in the precise locations and orientations of objects. Despite the accomplishments represented by SPAR, the requirement that the planner know in advance the whereabouts of all the objects has weighed heavily on us. We have given much thought to how sensors may be used optimally in a modern multi-sensor robotic assembly cell. We believe that by integrating, after further developments, the work we have reported in [18,19] with SPAR, a truly useful planner will emerge.

Much of today's planning research falls into one of two camps, the STRIPS family of planners (more generally the domain independent planners) [7,14,15,20,26,27,33,34,39,40,42], and the configuration space planners [11,22,25]. Neither of these approaches to planning is capable of producing complete assembly plans from high level specifications of assembly goals. The STRIPS family of planners, because of their high level treatment of the world, goals and actions, are not equipped to reason about geometric dependencies of actions or about uncertainties in the work cell. The configuration space planners are useful primarily for planning fine motion strategies, that is, these planners generally start with a known geometric goal state for a particular action (e.g. an insertion or grasp) and develop fine motion plans to achieve those goals. They are not equipped with descriptions of the preconditions for applying various robot actions, or with methods of introducing additional robot actions into plans to achieve unsatisfied preconditions. Furthermore, neither of these approaches has any mechanism for determining when or how to use sensing operations during plan execution to reduce uncertainties, or during the planning process to gain information about the initial world state (although configuration space planners do assume the existence of compliant moves which depend upon force/torque sensing, there is no planning involved in determining when to monitor the f/t sensor, since executing compliant motions is left to the robot controller).

One system which approaches the problem of planning geometric configurations for assembly actions is RAPT, described in [1,31,32]. However, this system is not really a planner in the true sense. RAPT's input is a symbolic description of the relationships which must hold in the goal state. Equations which define these relationships are symbolically manipulated to derive homogeneous transformations that represent the goal relationships between the manipulator and the work pieces to be manipulated. With this approach, the operator describes the goal configuration for each step in the assembly plan and RAPT determines the appropriate robot action, and the geometric parameters of that action. RAPT also does not know about preconditions of actions, or how to insert actions into plans if preconditions are not satisfied. For example, if a part which needs to be grasped is occluded by some other object in the work space, RAPT will not be able to plan the manipulations to remove the occluding object.

Attempts at dealing with world uncertainties also fall into two camps: planners which attempt to anticipate, and avoid errors [5,30], and error recovery planners which are invoked only after an execution error occurs [3,16,28,37,38]. The recovery planners do not actually consider the uncertainty in the world description. They start with a world description which is derived after the execution error occurs, and assume that this description is accurate. Pioneering work with error prevention planning is described by Brooks in [5]. The system described by Brooks is also not a true planner. Although it is capable of adding constraints to plan variables, his system is really a "plan checker." The input to Brooks' system is an assembly plan and a description of the world. The system determines whether or not the uncertainty in the world description is small enough to allow the plan to be executed successfully. If not, then sensing operations are added. If, however, the sensing operations cannot reduce the uncertainty to a sufficiently small level, the system exits, signaling that the plan was unsound.

Consideration of uncertainty in fine motion planning has been discussed in [6,10,13], but, as we have already stated, fine motion planning is merely one aspect of assembly planning. Also, a number of schemes for representing uncertainty in robotic systems have been described, for example [12,36], but these are not currently part of a planning system.

There are two systems, of which we are aware, whose scope is similar to the scope of SPAR. The first of these is TWAIN, developed by Lozano-Perez and Brooks [23]. TWAIN is a constraint posting planner which starts with task level plans and expands them to include gross motion plans, fine motion plans, grasping plans, and sensory operations to reduce uncertainty. The main difference between TWAIN and SPAR is that SPAR formulates its task level plans (which we refer to as operational plans) directly from assembly goals. By providing TWAIN with a task level plan, the system programmer has done the work of anticipating interacting goals and planning to achieve them. Furthermore, by supplying task level plans, a certain generality is lost to the system. For example, if the work cell's capabilities were upgraded to include an additional robot, new task level plans would have to be developed by the programmer. Clearly, a front end module could be added to TWAIN to formulate operational plans, but the control structure of SPAR allows interaction between the levels of planning, so that geometric level planning can directly influence the operational level planning. This type of interaction could not be accomplished by the simple addition of a front end operational planning module. A second difference is that TWAIN's constraints are all symbolic algebraic inequalities. In SPAR we use symbolic algebraic inequalities for uncertainty-reduction planning, but SPAR also uses a number of other constraints, for example a reachability constraint which is used to constrain the choice of grasping configurations to be physically realizable. By using these additional types of constraints, SPAR is able to avoid some of the complexities of constraint manipulation which were seen in TWAIN.

The Handey system, developed by Lozano-Perez, et. al. [24], is an integrated system which plans grasping and gross motions, and includes a sensory system to determine the initial world state. Handey also begins with a task level plan, which is comprised totally of MOVE commands of the form: MOVE object TO destination. One of Handey's main strengths is its ability to plan grasping operations when the objects are in cluttered environments. Handey is also able to plan "regrasping" operations when the object cannot initially be grasped in an acceptable configuration (this is discussed further in [41]). This planning is done using configuration space approaches. Handey does not have a mechanism for dealing with world uncertainties, nor does it have any method of postponing choices in the planning process (as do constraint posting planners).

The approach to planning which we describe in this paper was inspired by Chapman's work on the constraint posting method [7]. In the constraint posting method, the planner seeks to satisfy a goal by first examining all of the actions and constraints previously generated to see if the goal can be satisfied by the addition of a new constraint (where a constraint may be viewed as a specification or a restriction on an action). Only if this strategy fails, will a new action be added to the plan. Chapman's planner by itself would be incapable of handling the geometric and uncertainty caused complexities in a robot work cell, and its main virtue lies in the fact that it possesses some elegant theoretical properties, such as the property of completeness which implies that if a solution to a planning problem exists the planner would find it.

Our planner expands Chapman's constraint posting work by extending the planning beyond high level goals (which we refer to as operational goals) to include geometric and uncertainty-reduction goals. In order to plan with these additional goals, we have added a constraint manipulation system (CMS) which contains domain specific knowledge (including the kinematics of the robot, object models, and sensing operations). This domain knowledge is used by the CMS to determine whether or not constraints on such things as robot arm configurations can be satisfied. In order to organize the constraints, and to prevent the addition of an inconsistent constraint, the CMS maintains a constraint network. Plan variables which are constrained are represented by nodes in the network and binary constraints on the variables are represented by arcs connecting the corresponding nodes (where by a binary constraint, we mean a constraint on the relationship between two plan variables). The addition of a constraint to the plan corresponds either to the addition of an arc to the network (a new binary constraint), the addition of one or more nodes to the network (new plan variables), or the restriction of the label set of some node in the network (restricting the possible values which may be assigned to the plan variable). In each case, the CMS must verify that the new network will be consistent (i.e. that there is some assignment of labels to nodes which satisfies all constraint arcs) before adding the new constraint.

When designing a constraint posting planner, the degree to which constraint posting is used is an issue which must be considered. A pure constraint posting planner makes no variable instantiations until all of its goals have been satisfied, at which time the CMS is used to determine the variable instantiations which simultaneously satisfy all of the constraints in the constraint network. The advantage to this approach is that the planner is able to decrease the amount of backtracking by avoiding arbitrary choices which could lead to failure. The disadvantage to a pure constraint posting approach is that maintaining the constraint network can become more expensive than backtracking during planning. Therefore, in many cases a combination of constraint posting and backtracking is appropriate, the exact combination being determined by the complexities of the constraints and the cost of backtracking.

In SPAR, due to the complexities involved with the representation and evaluation of uncertainty-reduction goals, only the operational and geometric goals are satisfied using the constraint posting method (we will elaborate on these complexities in Section 4.3). Therefore, SPAR performs its planning in two phases. In the first phase constraint posting is used to construct a family of plans that satisfy all operational and geometric goals. In the second phase, specific plan instances (generated by instantiating the plan variables so that the constraint network is satisfied) are used as input for the uncertainty-reduction planning. We should note that the constraint posting paradigm is conceptually able to handle all three types of goals, however, for the reasons of complexity that we have just mentioned, it is not expedient to try and force uncertainty-reduction planning into the constraint posting mold. Furthermore, it would not be advantageous to abandon constraint posting for the operational and geometric planning, since the cost of maintaining the constraint network associated with these two types of goals is significantly less than the cost of implementing a backtracking search algorithm.

In the remainder of the paper, we will discuss in detail how SPAR creates assembly plans. In Section 2 we will give an overview of the system, including the top level search strategy used to satisfy system goals. In Section 3 we will describe the representations that SPAR uses for uncertainty in the world, plans, actions and goals. Section 4 describes how SPAR satisfies individual goals. This includes discussions on the satisfaction of high level operational goals, geometric goals (which define the robot configurations which will be used to execute the actions), and uncertainty-reduction goals (which express the maximum amount of uncertainty which can exist in the world description without causing execution failure). In Section 5, we discuss how SPAR represents and manipulates constraints. Section 6 brings the first sections of the paper together by presenting an example of how SPAR develops a plan for a basic assembly task. In Section 7, we discuss a number of issues which are useful in evaluating SPAR's effectiveness, for example completeness and correctness. Finally, Section 8 concludes the paper with a summary and allusions to future efforts.

2. PLANNING IN SPAR

In order to create complete assembly plans, we have extended the planning which is done in traditional constraint posting planners, such as those described in [7,42], to include both geometric planning and uncertainty-reduction planning. By geometric planning, we mean the planning which determines the actual geometric configurations which will be used during the assembly process. These configurations include the configurations of the manipulator, the positions in which parts are placed, and the grasping configurations which are used to manipulate objects. Uncertainty-reduction planning consists of first determining whether or not the uncertainty in the planner's description of the world (e.g. the possible error in part locations) is sufficiently small to allow plan execution to succeed. If the uncertainties are too large, then either sensing operations or manipulations are added to the plan in an attempt to reduce the uncertainty to an acceptable level. If this fails, verification actions and local recovery plans are added to the plan. These can be used during plan execution to monitor the robot's success and recover from possible run time errors. We call the resulting planner SPAR, for Simultaneous Planner for Assembly Robots, since all three levels of planning influence one another.

In SPAR, the planning process begins with a null plan and a set of goals which the user supplies. This null plan is then refined until all goals are satisfied. This occurs in two phases. First, a constraint posting approach is used to satisfy all operational and geometric goals. This results in a partial plan, consisting of an unordered set of plan actions and a set of constraints on how and when those actions are to be performed. The second phase of planning deals with the uncertainty-reduction goals. In this phase, SPAR creates specific plan instances (by instantiating the plan variables from the partial plan developed in the first phase so that all constraints are satisfied) and attempts to satisfy the uncertainty-reduction goals for the resulting actions.

The flow of control in the first phase of planning is a simple loop. On each iteration, SPAR selects one goal (either operational or geometric) and refines the current partial plan so that it satisfies that goal. This is done by either constraining the execution of some action which is already in the plan, or by introducing a new action into the plan. The first phase of planning terminates when there are no more pending operational or geometric goals.

In order to keep track of its progress during the first phase of planning, SPAR maintains two pending goal stacks (one each for operational and geometric goals) and a list of goals which have already been satisfied by the system. The planner determines which goal to consider by examining, in order, the operational and geometric goal stacks, and choosing the first pending goal that it finds. Initially, the pending goal stacks contain only those goals which are entered by a human user, and the list of satisfied goals is empty. During the planning process, any time an action is added to the plan, its operational and geometric preconditions are added to the appropriate pending goal stacks. The reason for the list of satisfied goals is that the introduction of a new action into the plan could possibly undo a previously satisfied goal. Therefore, when a new action is added to the plan, the planner checks each satisfied goal, and any which are undone by the new action are moved to the appropriate pending goal stack.

Moving goals which are undone by new actions back to a goal stack may seem an unwise strategy, and one which would lead the system into thrashing, as exemplified by constantly moving one goal back and forth from the satisfied goal list to one of the two goal stacks. Unfortunately, this is a property of almost all planning programs which deal with interacting goals and actions. Chapman has shown this to be true for TWEAK in his outcomes lemma [7], which states that each of three outcomes (termination in failure, termination in success, nontermination) is possible for some choice of domain and problem. It is for this reason that SPAR, like TWEAK, prefers adding constraints (which will never result in undoing a goal) to adding actions. It should be noted that, although there are ways to decrease the chances of nontermination by using meta-level control to detect looping, since planning using this constraint posting formalism is undecidable (this was also shown by Chapman), there is no way to completely eliminate the possibility that the planner will not halt.

Once all of the operational and geometric goals have been satisfied, SPAR moves to the second phase of planning, which deals with the uncertainty-reduction preconditions of the actions. As described earlier, in this phase of planning SPAR does not use the constraint posting approach. Instead, the uncertainty-reduction preconditions are considered for specific plan instances. In order to create these plan instances, SPAR invokes its CMS to find consistent solutions for the plan's constraint network. These solutions are then used to instantiate the variables in the plan actions. SPAR examines specific plan instances until it finds one in which all uncertainty-reduction goals can be satisfied. If no such instance can be found, it selects the instance which contained the fewest unsatisfied uncertainty-reduction goals as being the best plan.

In the second phase of planning, for a particular plan instance, SPAR iteratively examines each action in the plan instance to determine whether that action's uncertainty-reduction preconditions can be satisfied given the uncertainty in the world state just prior to the action's execution. For the first action in the plan, the uncertainty in the world state is simply the uncertainty in the initial world state. For the n^{th} action in the plan, the uncertainty in the world state is determined by propagating the initial uncertainties forward through the first n-1 actions in the plan instance. If the uncertainty in the world state is too large to satisfy an action's uncertainty-reduction preconditions, SPAR attempts to add either sensing or manipulations to the plan to reduce the uncertainty to an acceptable level. If this fails, SPAR adds sensing actions to the plan instance, which will be used at the time of plan execution to verify the success of the action. Local recovery plans are also added, based on the types of error which are likely to occur given the particular uncertainty-reduction goal which was not satisfied.

In both phases of planning, the search strategy used is depth first search. No great loss is suffered by using depth first search in the first phase of planning, since the primary advantage to the constraint posting approach is that it allows the planner to delay making choices until absolutely necessary, which in turn limits the amount of backtracking required. An additional reason for the choice of depth first search instead of breadth first search (which applies to both phases of planning) is that plans in SPAR can become very large and complex, which means that the storage associated with breadth first search methods can grow to be prohibitively large.

Fig. 1 shows a block diagram of SPAR. To the left are the goal stacks and set of satisfied goals. SPAR uses these to keep track of its planning progress. At the top, enclosed by a dashed box, are the templates which are used to represent actions, a set of rules for instantiating those templates, a set of actions to be used to reduce uncertainty in the world, and a set of procedures which are used to construct the uncertainty-reduction preconditions for actions in the plan. To the right, enclosed by a dashed box, is the constraint system. This includes the actual CMS, a number of domain dependent modules (e.g. upper and lower bounding routines, an algebraic simplifier, inverse kinematics of the robot), and a constraint network which is used to organize the plan's constraints. Finally, at the bottom of the figure are the verification sensory operations and local recovery plans, which are used when uncertainty-reduction goals cannot be satisfied, as well as the set of actions which are currently in the plan.

3. REPRESENTATIONAL ISSUES IN SPAR

One of the important issues which must be addressed when designing a planning system is the choice of representation schemes which will be used. These representations determine the power that the planner will have, in terms of its ability to adequately model the world and the possible actions which can be performed to alter the world. In this section, we will describe how SPAR represents actions, uncertainty, plans, and goals.

3.1. Representation of Actions

Currently, SPAR plans with three actions: pickup, putdown and assemble.* These are represented by the action templates shown in Figs. 2-4. Each action template has the following components:

Action id: An identifier which SPAR uses to reference a particular instance of the action.

*There is only a limited repertoire of actions that can be carried out by a single robot arm and the three listed here represent those that are used most often. Actions like threading and fixturing could be considered to be more specialized forms of the assemble action presented here, the specialized forms being obtained by the addition of more geometrical and uncertainty-reduction constraints.

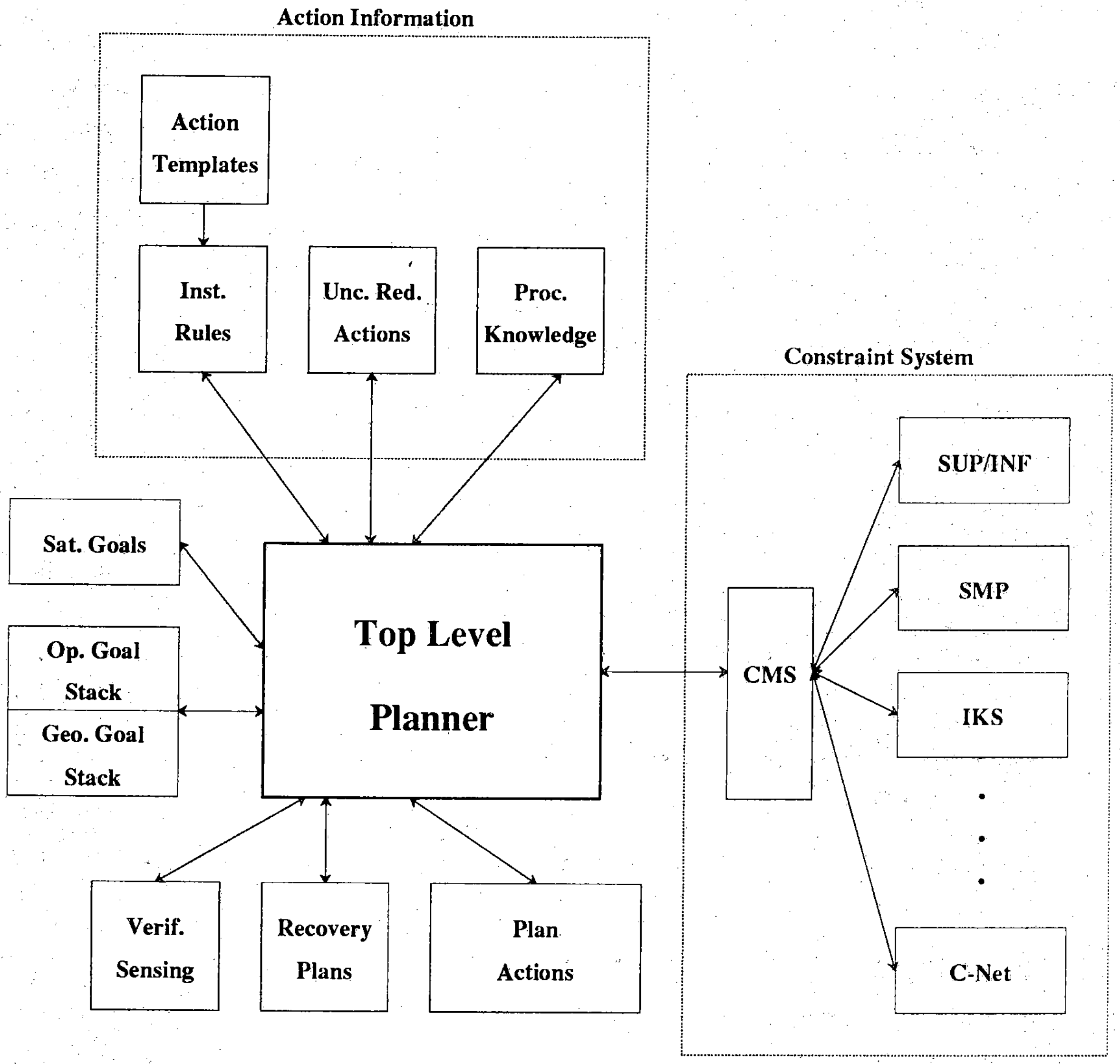

Figure 1: Block diagram of SPAR.

```
action-id:     ActionId,

action:        pickup(Object,Grasp),

preconditions:

     operational:

          op(G1,ActionId, gripper(open))
          op(G2,ActionId, part_location(Object,Pos))

     geometric:

          geo(G3,ActionId,
               reachable(Grasp,Pos),
               part_location(Object,Pos))

     uncertainty-reduction:
```

$$0 < [0,1,0,0]\, P_1^{-1}\, C_1$$
$$0 > [0,1,0,0]\, P_1^{-1}\, C_2$$
$$0 < [0,1,0,0]\, P_2^{-1}\, C_1$$
$$0 > [0,1,0,0]\, P_2^{-1}\, C_2$$

```
add-list:

     holding(Object,Grasp)
     part_location_unc(Object,NewUnc)
     gripper(closed)

delete-list:

     part_location(Object,Pos)
     part_location_unc(Object,OldUnc)
     gripper(open)
```

Figure 2: The action template for the pickup action.

```
action-id:      ActionId

action:         putdown(Obj,Pos)

preconditions:
        operational:
                op(G1,ActionId,holding(Obj,Grasp))

        geometric:
                geo(G2,ActionId,
                        reachable(Grasp,Pos),
                        holding(Obj,Grasp))

        uncertainty-reduction:
                nil

add-list:
        part_location(Obj,Pos)
        part_location_unc(Obj,NewUnc)
        gripper(open)

delete-list:
        gripper(closed)
        part_location_unc(Obj,OldUnc)
        holding(Obj,Grasp)
```

Figure 3: The action template for the putdown action.

```
action-id:      ActionId,

action:         assemble(Obj1,Obj2,[M1,M2],Transform,Mvector),

preconditions:

        operational:
                op(G1,ActionId,holding(Obj1,Grasp))
                op(G2,ActionId,part_location(Obj2,Pos))

        geometric:
                geo(G3,ActionId,
                        member(Grasp,GraspList),
                        holding(Obj1,Grasp))
                geo(G4,ActionId,
                        in_position_class(Pos,PositionList),
                        part_location(Obj2,Pos))
                geo(G5,ActionId,
                        mate_reachable(Grasp,Pos,Transform),
                        holding(Obj1,Grasp),
                        part_location(Obj2,Pos))

        uncertainty-reduction:
                $(P_{1x} - P_{2x})^2 + (P_{1y} - P_{2y})^2 + (P_{1z} - P_{2z})^2 - (R_1 - R_2) > 0$

add-list:
        assembled(Obj1,Obj2,[M1,M2],Transform,Mvector)
        gripper(open)

delete-list:
        gripper(closed),
        holding(Obj1,Grasp)
```

Figure 4: The action template for the assemble action.

Action:	The name of the action, and its arguments.
Preconditions:	The operational geometric, and uncertainty-reduction preconditions which must be met prior to the execution of the action.
Add list:	A list of conditions which will be true in the world after the execution of the action.
Delete list:	A list of conditions which will no longer be true in the world after the execution of the action.

Two points should be made about these figures. First, rather than present Prolog code (SPAR's current implementation language is Prolog), we have chosen a language independent representation of the actions for the purposes of illustration. In general, we will avoid actual Prolog code whenever possible. Second, in the representations of the actions, the uncertainty-reduction goals, as well as the elements of the add and delete lists which affect the propagation of uncertainty, are presented in a declarative style in the interest of clarity. In the actual implementation, these are encoded as procedures which are attached to the actions and invoked when needed during the second phase of planning. In other words, when actions are first invoked via a Prolog clause the uncertainty-reduction parts shown in the figures are not invoked, because they are really not a part of the clause. Therefore, the variables such as P_1 and C_1 do not exist at the instant the action in Fig. 2 is first invoked. It is only during the uncertainty reduction phase -- a phase that will be explained in detail in Section 4.3 -- that procedures are initiated to determine the instantiations for those variables.

When SPAR determines the need to add an action to the plan, it instantiates the template for that action so that it will accomplish the particular goal which caused the action to be added. This consists of first instantiating the various identifiers in the action to unique labels (e.g. the ActionId the Gi's), and then either instantiating or constraining the plan variables in the action so that it achieves the goal. For example, when SPAR adds an assemble action to the plan, the variables Obj1 and Obj2 are instantiated to the names of the two objects to be assembled. The variable GraspList (in the first geometric precondition) is instantiated to be the set of grasps which does not obscure the mating features of Obj1.

SPAR uses a set of rules to determine the proper variable instantiations for an action template, given the goal which the action is to achieve. Fig. 5 shows an example of the rule which instantiates a pickup action template to achieve the goal holding(Object,Grasp). The Prolog implementation of the rule is shown in Fig. 6. (Note that the uncertainty-reduction preconditions do not appear in the instantiated template, since they are actually encoded as procedures.) By using this approach to instantiating action templates, SPAR is able to use a small set of generic robot operations and instantiate these to specific actions based on the objects which will be manipulated by those actions. One alternative to using a rule base for selecting and instantiating actions would be to search the list of possible actions for an action whose add list contained an element which could be unified with the goal. In general, this type of search

```
If the Goal is "holding(Object,Grasp)" then:

        Generate a unique symbol for Grasp

        Set GraspList to the list of grasping configurations for Object

        Set Initial Constraint List to contain "member(Grasp,GraspList)"

        Generate unique symbols for: ActionId, G1, G2, G3

        Set Template =
                action( ActionId,
                        pickup(Object,Grasp),
                        preconditions(
                                [op(G1,ActionId, gripper(open)),
                                 op(G2,ActionId, part_location(Object,Pos))],

                                [geo(G3,ActionId,
                                        reachable(Grasp,Pos),
                                        part_location(Object,Pos))]),

                        addlist([holding(Object,Grasp),
                                 gripper(closed)]),

                        dellist([part_location(Object,Pos),
                                 gripper(open)]))
```

Figure 5: Rule to instantiate an action template.

```
instantiate_establishing_action(holding(Object,Grasp),Action,Constraints) :-
        action_template(pickup(Object,Grasp),Action),
        get_all_grasps(Object,GraspList),
        Constraints = [member(Grasp,GraspList)].


action_template(pickup(Object,Grasp),Template) :-

        gensym(action_,ID),
        gensym(goal_,G1),
        gensym(goal_,G2),
        gensym(goal_,G3),
        gensym(grasp_,Grasp),

        Template =
                action( ActionId,
                        pickup(Object,Grasp),
                        preconditions(
                                [op(G1,ActionId, gripper(open)),
                                 op(G2,ActionId, part_location(Object,Pos))],

                                [geo(G3,ActionId,
                                        reachable(Grasp,Pos),
                                        part_location(Object,Pos))]),

                        addlist([holding(Object,Grasp),
                                gripper(closed)]),

                        dellist([part_location(Object,Pos),
                                gripper(open)]))
```

Figure 6: The Prolog rule which is used to invoke action template instantiation and then to instantiate the initial constraints for the pickup action.

and unification is less efficient than our approach.

In some cases, when an action is added to the plan, an initial set of constraints on the plan variables is also added. For example, the rule shown in Fig. 5 specifies the initial constraint that Grasp be one of the possible grasps for Object. (We will discuss how grasps, stable poses, etc. are represented in the Section 5.) This amounts to assigning an initial label set to a node in the constraint network (this will also be discussed further in Section 5). When initial constraints are added, there is no need to invoke the CMS to see if they are consistent with the current constraint network, since the variables which will be constrained by these initial constraints did not previously exist in the plan, and thus did not occur in the constraint network.

3.2. The Representation of Uncertainty in SPAR

In order to create assembly plans which are to be executed in an uncertain environment, SPAR must have a suitable representation for the uncertainty in its world description, an understanding of how much uncertainty in that description can be tolerated before an action can no longer be guaranteed to succeed, and a knowledge of how the various assembly actions affect the uncertainty in the world description. In this section, we will address each of these three issues. First, we discuss how SPAR represents uncertainties in the position of the manipulator, the positions of objects which are resting on the work table, and the positions of objects which are held in the manipulator. Once we have described the representations for position uncertainties, we will derive the uncertainty-reduction preconditions for the assemble and pickup actions. Finally, we will describe how the actions affect the uncertainty in the world description.

3.2.1. Representing Uncertain Quantities

In our current implementation of SPAR, we have chosen to limit the number of quantities which are considered to be uncertain. For an object resting on the work table, the X,Y,Z location of the object (i.e. the object's displacement) and the rotation about an axis through the origin of the object's local frame and perpendicular to the table are considered uncertain. This is illustrated in Fig. 7. This choice reflects our assumption that objects resting on the work table will be in a stable pose, which fixes two rotational degrees of freedom of the object (this assumption will be discussed further in Section 5). For the manipulator, we consider the X,Y,Z location of the tool center, and the rotation about the Z axis of the manipulator's local frame to be uncertain. The local frame of the manipulator is illustrated in Fig. 8.

All uncertainties in SPAR are expressed in terms of uncertainty variables. The possible values for an uncertainty variable are defined using bounded sets. For example, the uncertainty in the X coordinate of an object might be represented by the uncertainty variable ΔX, where $-0.5 < \Delta X < 0.5$. We represent the uncertainty in the position of an object by a homogeneous transformation matrix whose entries are expressed in terms of uncertainty variables. By combining the ideal position of an object (i.e. the position of the object if all uncertainty is eliminated) with the uncertainty in that position, we obtain the possible position of an object.

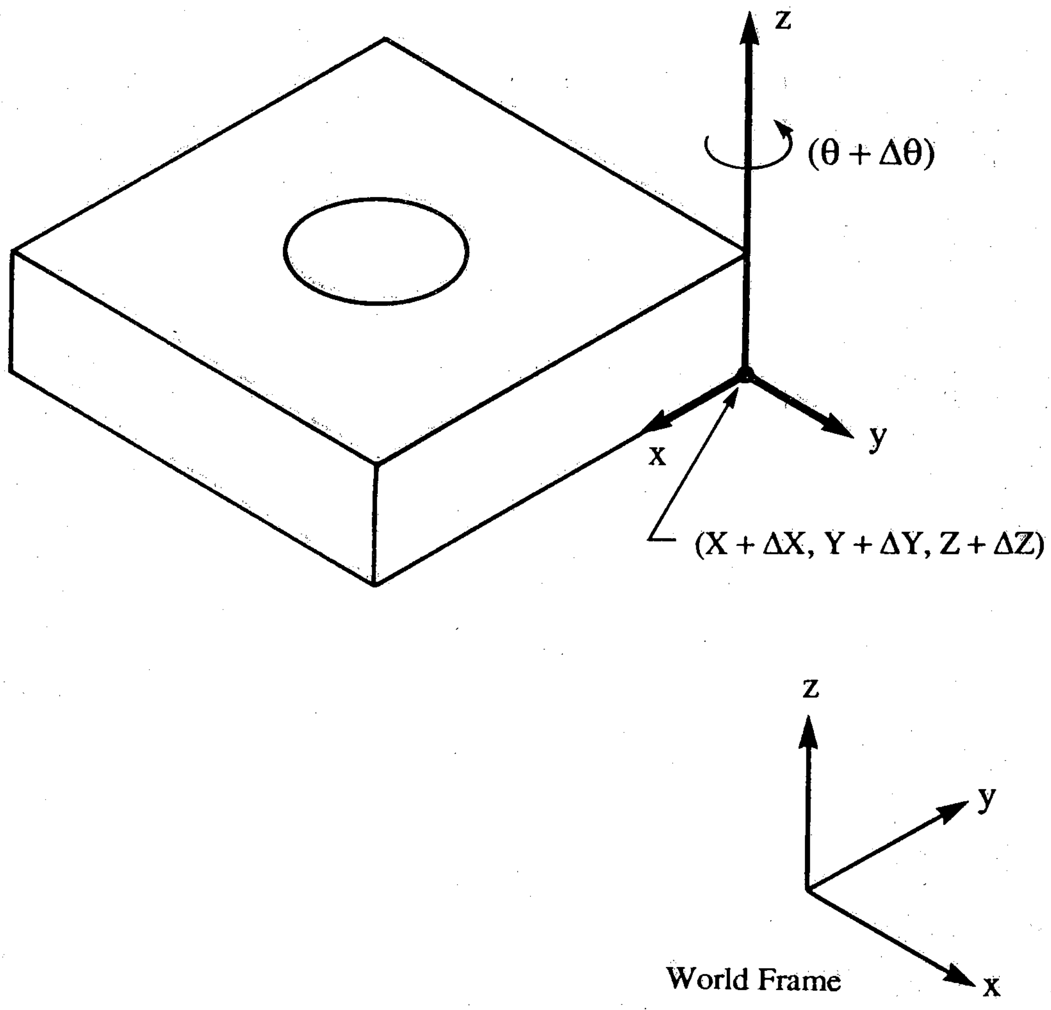
z
$(\theta + \Delta\theta)$
x
y
$(X + \Delta X, Y + \Delta Y, Z + \Delta Z)$
z
y
World Frame
x

Figure 7: Possible uncertainties in the position of an object on the work table.

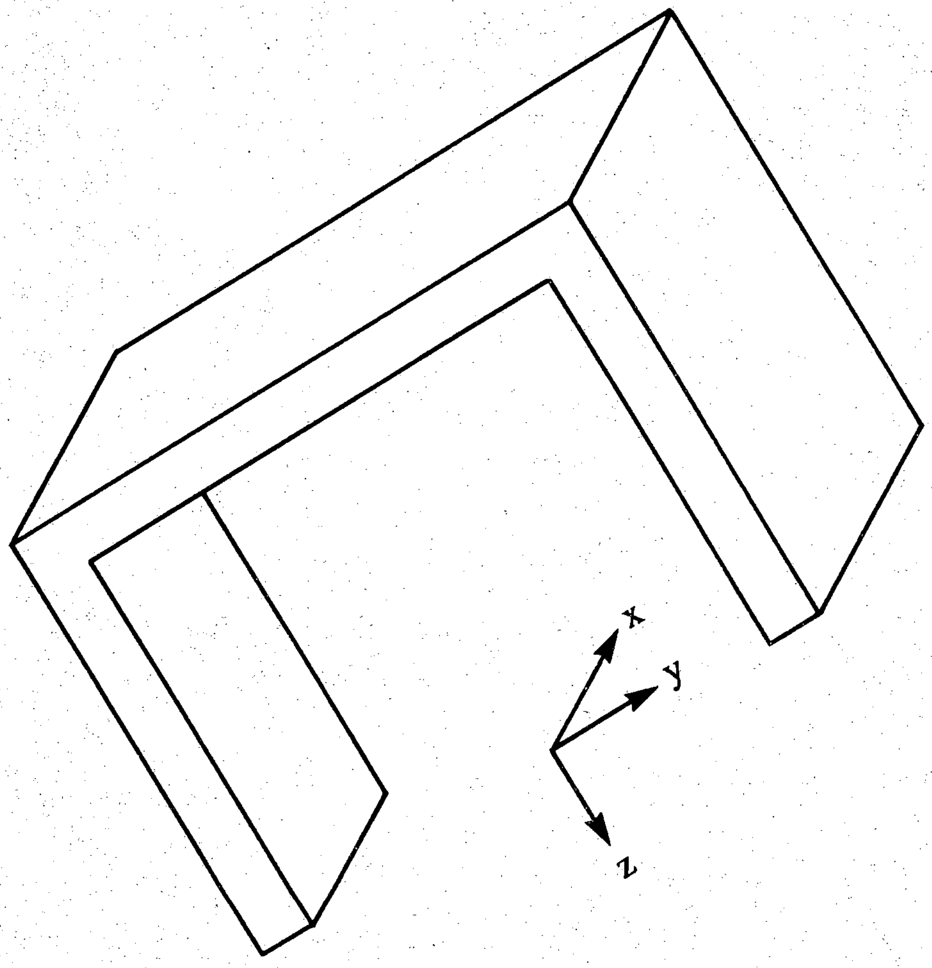

Figure 8: The manipulator's local coordinate frame.

This possible position will be a homogeneous transformation matrix, with some or all of its entries expressed in terms of uncertainty variables. Any matrix which can be obtained by substituting valid values for the uncertainty variables will represent one possible position of the object.

Given this formalism, and given that we consider the uncertainty in the manipulator's position to be limited to the X,Y,Z location of the tool center and the rotation about the manipulator's local Z axis, we can define the transformation which represents the uncertainty in the position of the manipulator relative to the manipulator's local frame to be:

$$T_{\Delta M} = \begin{bmatrix} \cos(\Delta\theta_g) & -\sin(\Delta\theta_g) & 0 & \Delta X_g \\ \sin(\Delta\theta_g) & \cos(\Delta\theta_g) & 0 & \Delta Y_g \\ 0 & 0 & 1 & \Delta Z_g \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Again, note that the values ΔX_g, ΔY_g, ΔZ_g, and $\Delta\theta_g$ are bounded symbolic variables. Therefore, the matrix $T_{\Delta M}$ represents all of the transformations which could be obtained by substituting valid numerical values into the matrix in place of the symbolic variables. The bounds on these variables are stored in SPAR's database, and retrieved when needed.

Given that $T_{\Delta M}$ represents the uncertainty in the manipulator's position relative to the manipulator's own local frame, we can compute the possible position of the manipulator (i.e. the combination of ideal position and possible error) using the composition:

$$T_{M+\Delta} = T_M \, T_{\Delta M} \qquad (1)$$

where T_M represents the ideal position of the manipulator.

The expression for the possible position of an object resting on the work table is a bit more complicated, due to the rotational component in the uncertainty. In particular, the axis of this rotation is not defined by the local frame of the object or by the world frame, but by the world Z axis, translated to the origin of the object's local frame. To deal with this, we will consider the uncertainty in the displacement of the object and the rotational uncertainty separately. For the displacement, let the transformation $Tr_{\Delta O}$ be a transformation which defines the uncertainty in the X,Y,Z location of the object relative to the world coordinate frame:

$$Tr_{\Delta O} = \begin{bmatrix} 1 & 0 & 0 & \Delta X_O \\ 0 & 1 & 0 & \Delta Y_O \\ 0 & 0 & 1 & \Delta Z_O \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, let the transformation Tr_O represent the ideal X,Y,Z position of the object. We obtain the possible displacement of the object's local frame by combining the two:

$$Tr_{O+\Delta} = Tr_{\Delta O} \, Tr_O$$

Now, since the rotational uncertainty is about the world Z axis translated to the origin of the object's local frame, it can be represented by post-multiplying the possible object displacement by a rotation about the Z axis:

$$T_{O+\Delta} = Tr_{\Delta O}\, Tr_O\, R_{\Delta O}$$

where:

$$R_{\Delta O} = \begin{bmatrix} \cos(\Delta\theta_O) & -\sin(\Delta\theta_O) & 0 & 0 \\ \sin(\Delta\theta_O) & \cos(\Delta\theta_O) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Finally, by defining the matrix R_O to denote the ideal orientation of the object, we obtain the possible position of the object (which includes both displacement and rotation uncertainties) as:

$$T_{O+\Delta} = Tr_{\Delta O}\, Tr_O\, R_{\Delta O}\, R_O \qquad (2)$$

When an object is in the grasp of the manipulator, its position uncertainty is expressed in terms of the uncertainty in the manipulator's position (this will be explained when we describe how the pickup action affects the uncertainty in the world description). We will use T_G to represent the transformation which specifies the grasp (i.e. T_G specifies the transformation from the object's local frame to the local frame of the manipulator). Therefore, neglecting uncertainty, we have:

$$T_M = T_O\, T_G \qquad (3)$$

and consequently:

$$T_O = T_M\, T_G^{-1}$$

Using this equation, and remembering that the uncertainty in a grasped object's position is defined in terms of the position of the manipulator, for an object in the manipulator's grasp, with the uncertainty transformation $T_{\Delta O}$, the possible position of the object is:

$$T_{O+\Delta} = T_M\, T_{\Delta O}\, T_G^{-1} \qquad (4)$$

Note that $T_{\Delta O}$ is not the same as $Tr_{\Delta O}$. The former describes the uncertainties remaining after the object has been grasped, and can only be computed by analyzing the interaction of the manipulator with the object, as for example illustrated by our derivation in Section 3.2.3.

3.2.2. Derivation of Uncertainty-Reduction Goals

Currently, SPAR has uncertainty-reduction preconditions for both the pickup and assemble actions. We have not developed uncertainty-reduction preconditions for the putdown action, since placing an object on the table can generally be guaranteed to succeed if the position where the object is to be placed is sufficiently isolated from the other items in the work space.

The uncertainty-reduction preconditions for the pickup action are derived by examining the possible locations of the manipulator fingers, and their relationship to the possible locations of the contact points on the object to be grasped (i.e. the points on the object which the fingers will contact in the grasping operation). These preconditions should guarantee that the two contact points will lie between the fingers of the manipulator, even when worst case uncertainties occur. To express this, we first derive the possible local coordinate frames for each finger. We then derive the possible locations of the contact points on the object. Finally, we transform the possible contact points so that they are expressed in the local finger frames, and check that they each lie between the fingers.

To find the possible local frames of the manipulator fingers, we find the possible location of the manipulator's local frame and perform a translation of $\pm\frac{1}{2}W_m$ along the Y axis of that frame (where W_m is the distance between the two fingers). This is illustrated in Fig. 9. Using Eqs. 1 and 3, we find:

$$P_1 = T_O\, T_G\, T_{\Delta M}\, \mathrm{trans}(0, -\tfrac{1}{2}W_m, 0)$$

$$P_2 = T_O\, T_G\, T_{\Delta M}\, \mathrm{trans}(0, +\tfrac{1}{2}W_m, 0)$$

where trans(X,Y,Z) indicates a transformation of the form:

$$\mathrm{trans}(X,Y,Z) = \begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_O is the position of the object to be grasped. T_G is the gripping transformation (i.e. the transformation which expresses the position of the manipulator's coordinate frame relative to the object's local frame), and $T_{\Delta M}$ is the uncertainty in the manipulator position.

Note that although there will be uncertainty in the position of the object to be grasped, this uncertainty does not affect the calculation the fingers' possible local coordinate frames. The reason for this is that at the time of plan execution, the ideal position of the object, in conjunction with the grasping transformation, will be used to determine the manipulator configuration, T_M. The uncertainty in the location of the object will only affect the expressions for the two contact points on the object.

In order to determine the two possible contact points, C_1 and C_2 we first find the possible position of the object. Relative to the object's local frame, the contact points are obtained using the grasping transformation, T_G, in conjunction with a translation along the Y axis of the manipulator frame (i.e. the axis which defines the direction of finger opening and closing). Using Eq. 2, we find:

$$C_1 = Tr_{\Delta O}\, Tr_O\, R_{\Delta O}\, R_O\, T_G\, \mathrm{trans}(0, +\tfrac{1}{2}W_g, 0)[0,0,0,1]^t$$

$$C_2 = Tr_{\Delta O}\, Tr_O\, R_{\Delta O}\, R_O\, T_G\, \mathrm{trans}(0, -\tfrac{1}{2}W_g, 0)[0,0,0,1]^t$$

where W_g is the width of the object at the grasp point. Note that we are not interested in the

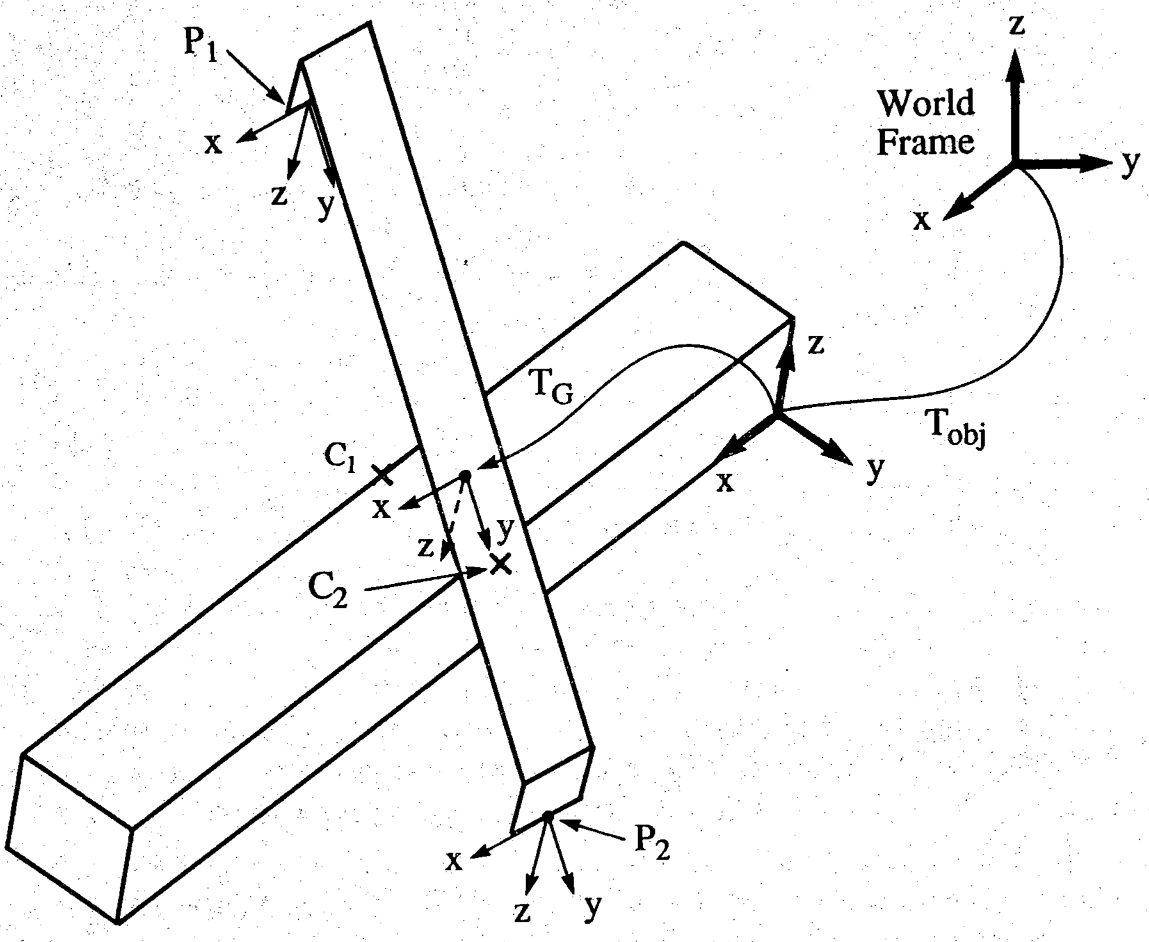

Figure 9: Possible finger coordinate frames, and contact points for the pickup action.

coordinate axes at the contact points, only the displacement.

In order to see if the contact points lie between the fingers, we transform the locations of C_1 and C_2 to be defined in terms of the coordinate frames P_1 and P_2, and check to see that the Y-components of these locations are on the positive Y axis for P_1 and on the negative Y-axis for P_2, for all possible values of the uncertainty variables. Therefore, the four uncertainty-reduction preconditions for the pickup action are:

$$0 < [0,1,0,0]\, P_1^{-1}\, C_1, \quad 0 < [0,1,0,0]\, P_1^{-1}\, C_2$$

and

$$0 > [0,1,0,0]\, P_2^{-1}\, C_1, \quad 0 > [0,1,0,0]\, P_2^{-1}\, C_2$$

Again, note that all of the matrix multiplications shown above must be performed symbolically. This is because many of the entries in the matrices will be expressed in terms of uncertainty variables which do not have specific numeric values.

For the uncertainty-reduction precondition of the assemble action, we currently only consider assembly operations which cause an object with a round peg to be mated to an object with a round hole, as shown in Fig. 10. Therefore, the precondition can be expressed in terms of the center point of the peg, C_p, and the center of the hole, C_h. In particular, if the distance between C_p and C_h is less than the difference in the radii of the hole and peg, the assemble action will succeed.

As we discussed in Section 3.1, the assemble action is expressed as:

assemble(P,H,Mfeatures,T_a,Vm)

where P (for peg) is the held object, H (for hole) is the object resting on the table, and T_a is the goal position of P relative to H's coordinate frame. In addition to T_a, we define F_p to be the translation vector from the local frame of P to the point centered at the tip of the peg, and F_h to be the translation vector from the local frame of H to the center of the hole. Fig. 10 illustrates these relationships, however, for clarity, we have depicted C_h to be at the top of the hole, and T_a to be the transformation from H's frame to P's frame just prior to the assembly action. In the actual implementation, C_h is at the base of the hole, and T_a is the transformation from H's frame to P's frame after the completion of the assembly action.

The possible position of the center of the hole can be found simply by applying the transformation F_h to the possible position of H. Remembering Eq. 2, we have:

$$C_h = Tr_{\Delta H}\, Tr_H\, R_{\Delta H}\, R_H\, F_h$$

The possible position of the center of the tip of the peg is a bit more complex. From Eq. 4, since P is held in the manipulator, its possible position is:

$$T_{P+\Delta} = T_M\, T_{\Delta P}\, T_G^{-1} \qquad (5)$$

(where T_M is the ideal manipulator position). However, the destination position P is defined in

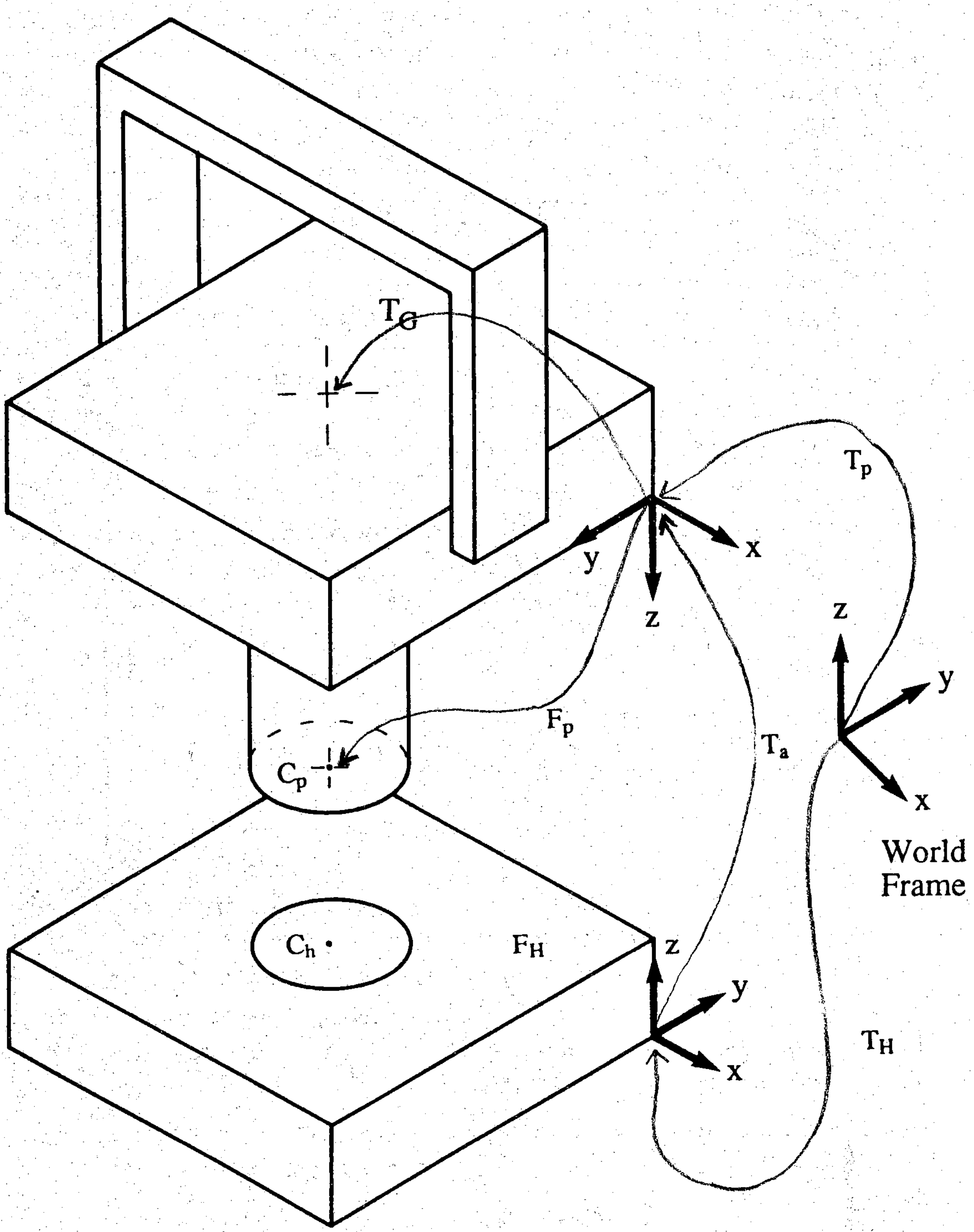

Figure 10: The parameters of the uncertainty-reduction precondition for the assemble action.

terms of T_a and the position of H to be:

$$T_P = T_H T_a$$

Recalling Eq. 3:

$$T_M = T_P T_G$$

we find that:

$$T_M = T_H T_a T_G \tag{6}$$

Therefore, combining Eqs. 5 and 6, the possible position of P is:

$$T_{P+\Delta} = T_H T_a T_G T_{\Delta P} T_G^{-1}$$

and by applying the translation F_p, we obtain:

$$C_p = T_H T_a T_G T_{\Delta P} T_G^{-1} F_p$$

The uncertainty constraint is expressed as

$$| C_h - C_p | - (R_h - R_p) > 0$$

or:

$$(C_{hx} - C_{px})^2 + (C_{hy} - C_{py})^2 + (C_{hz} - C_{pz})^2 - (R_h - R_p) > 0$$

3.2.3. The Propagation of Uncertainty by Actions

Currently, both the pickup and putdown actions change the uncertainty in the world description. At this time, the assemble action does not change the uncertainty in the world description.

As we discussed in Section 3.1, the pickup action is represented by:

pickup(Object,Grasp)

In general this action has the effect of reducing the uncertainty in the position of Object. This is because the new uncertainty in Object's position will be defined in terms of the manipulator uncertainty, which is normally less than the uncertainty in positions which are determined by the sensing system. Specifically, the pickup action has the effect of transforming Object's displacement uncertainty into the manipulator coordinate frame, and then reducing the Y component of this uncertainty to the uncertainty in the Y component of the manipulator. The pickup action also reduces the uncertainty in Object's orientation to be equal to the uncertainty in the orientation of the manipulator.

In order to represent this, let $Tr_{\Delta O}$ be the displacement uncertainty in Object's position just prior to the execution of the pickup action. Therefore, as described in Section 3.2.1, this uncertainty is defined relative to the world coordinate frame. What we need, is to obtain a displacement error $Tr'_{\Delta O}$ such that

$$R_M \, Tr'_{\Delta O} = Tr_{\Delta O}$$

where R_M is the transformation which represents only the orientation of the manipulator (i.e. it has a null displacement vector). In other words, $Tr'_{\Delta O}$ expresses the displacement error relative to the manipulator frame after the object is grasped. If we define R_O to be the transformation that represents the orientation of Object, and R_G to be the transformation which represents the orientation of the manipulator relative to the local frame of Object (i.e. the rotational part of T_G), we find:

$$Tr'_{\Delta O} = (R_O \, R_G)^{-1} \, Tr_{\Delta O}$$

since we know that

$$R_M = R_O \, R_G$$

and therefore,

$$R_M \, [(R_O \, R_G)^{-1} \, Tr_{\Delta O}] = Tr_{\Delta O}$$

Now, we define the vector that represents the uncertainty in Object's displacement relative to the manipulator frame by:

$$[Dx, Dy, Dz, 1]^t = (R_O \, R_G)^{-1} \, Tr_{\Delta O} \, [0,0,0,1]^t$$

Finally, by combining this displacement with the uncertainty in the position of the manipulator, we obtain:

$$T_{\Delta O} = \begin{bmatrix} \cos(\Delta\theta_g) & \sin(\Delta\theta_g) & 0 & Dx + \Delta X_g \\ \sin(\Delta\theta_g) & \cos(\Delta\theta_g) & 0 & \Delta Y_g \\ 0 & 0 & 1 & Dz + \Delta Z_g \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that the uncertainty in the Y component of the displacement uncertainty has been limited to the uncertainty in the Y component of the location of the manipulator's tool center. Further, note that the rotational uncertainty is the same as the rotational uncertainty in the orientation of the manipulator.

The putdown action, which can be represented by:

putdown(Object,Position)

affects the uncertainty in Object's position just the opposite of the pickup action. In particular, the position error is transformed from the manipulator's local frame to the world coordinate frame, and the uncertainty in the manipulator's orientation is transformed to reflect the uncertainty in Object's orientation about an axis through the origin of its local frame and perpendicular to the work table.

The vector which represents the uncertainty in Object's displacement is:

$$[Dx, Dy, Dz, 1]^t = (R_O \, R_G) \, Tr_{\Delta O} \, [0,0,0,1]^t$$

where R_O and R_G are defined as above, and $Tr_{\Delta O}$ is the uncertainty in Object's position, expressed in terms of the uncertainty in the position of the manipulator.

For the orientation, we have:

$$\Delta\theta = Z \times \Delta\theta_g$$

where Z is the projection of the Z axis of the frame defined by $(R_O\, R_G)$ onto the world Z axis. This effectively truncates two degrees of freedom of the orientation error, leaving only the orientation about an axis perpendicular to the work table.

The final uncertainty transformation is:

$$T_{\Delta O} = \begin{bmatrix} \cos(\Delta\theta) & \sin(\Delta\theta) & 0 & Dx \\ \sin(\Delta\theta) & \cos(\Delta\theta) & 0 & Dy \\ 0 & 0 & 1 & Dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.3. Representation of Plans

As described earlier, SPAR does its planning in two phases. During the first phase, a constraint posting approach is used to satisfy operational and geometric goals, and during the second phase, specific plan instances are examined to find plans which satisfy uncertainty-reduction goals. Clearly, the representation of plans must be different for these two phases.

The plans developed by SPAR during the first phase of planning are not simple linear sequences of actions. Instead, these plans consist of an unordered set of actions and a separate set of constraints on how and when those actions are to be executed. These constraints are stored in SPAR's constraint network. The constraints on how actions are executed are actually constraints on possible values which may be assigned to plan variables. For example, if the action is to grasp an object, constraints on the variable used to indicate the grasping configuration will effectively constrain how the grasping action is performed. Table 1 lists the constraints SPAR currently uses in the first phase of planning.

With this type of representation, a plan developed by SPAR during the first phase of planning actually corresponds to a family of plans. A specific plan instance is derived by finding a consistent instantiation for the plan variables (i.e. a set of values for the plan variables which satisfies the constraints in the constraint network), and performing that instantiation on the plan actions.

In the second phase of planning, SPAR uses specific plan instances, which are augmented to contain verification sensory actions, local recovery plans, and an error count. The verification sensory actions and local recovery plans are added when the uncertainty-reduction preconditions for an action cannot be satisfied, and are stored in separate lists. The error count is simply an integer which is incremented each time the uncertainty-reduction goals for an action in the plan instance cannot be satisfied. This error count is used to determine which plan instance has the

Table 1: The constraints which are used in the first phase of planning.

prior_to(Action1,Action2):	Action1 must be executed prior to Action2.
reachable(Grasp,Position):	Grasp is a grasping configuration which must be physically realizable when grasping an object located in Position.
mate_reachable(Grasp,P,T):	P defines a fixed coordinate frame. T defines the destination coordinate frame of the object which is held in the manipulator, relative to P. Grasp is a grasping configuration which must be physically realizable given T and P.
member(Item,List):	Item must be chosen from List.
in_position_class(Position,PositionList):	Position must be an element of PositionList.

greatest chance of success.

3.4. Representation of Goals

Goals in SPAR have three relevant attributes: a type (either operational, geometric or uncertainty-reduction) a condition which must be satisfied (i.e. the actual goal) and an action identifier. The action identifier is used to indicate when the goal must be satisfied, in particular, that it must be satisfied prior to the execution of the action specified by the action identifier. We will use the terms goal and precondition to refer to either the condition part of the goal or to the entire structure (which includes the action id and goal id). Which of these is meant should be clear by the context.

SPAR's operational goals are similar to the high level goals used in traditional domain independent planners (e.g. STRIPS or TWEAK). One difference is our inclusion of plan variables which can be used to link the operational and geometric goals. For example, one operational precondition of the assemble action is:

```
op(G1, ActionId, holding(Obj1, Grasp))
```

The plan variable Grasp is not used in the operational planning, but serves the purpose of linking the operational and geometric planning. The variable ActionId is used to indicate the time at which the goal must be satisfied. In particular, it must be satisfied just prior to the execution of the action whose action identifier is ActionId.

Geometric goals are slightly more complex, with two main components. The first is a geometric constraint and the second is a set of operational goals. The meaning of this pair is that the planner is to establish the operational goals in such a way that the geometric constraint is satisfied. For example, one geometric precondition of the putdown action shown in Fig. 3 is:

```
geo(G2, ActionId,
        reachable(Grasp, Pos),
        holding(Obj, Grasp))
```

In order to satisfy this condition, the planner searches for an action in the plan which achieves the goal holding(Obj,Grasp) and then attempts to constrain the execution of that action so that the condition reachable(Grasp,Pos) is satisfied. In order to determine whether or not Pos is reachable, the CMS is invoked, which in turn invokes the kinematic routines for the robot. The variables G2 and ActionId are instantiated when the putdown action is added to the plan. The variable G2 is instantiated to a label used to reference to this specific precondition, and the variable ActionId is instantiated to the label which is assigned to the putdown action.

The representation of uncertainty and the uncertainty-reduction goals was discussed in detail in Section 3. Uncertain quantities are represented by symbolic variables. These variables have associated upper and lower bounds. The uncertainty in the position of an object is represented by a homogeneous transformation matrix, some or all of whose entries are defined in terms of the uncertainty variables. By combining the ideal position of an object with the uncertainty in that object's position, a transformation is derived which defines the range of possible positions of the object. The uncertainty-reduction preconditions of an action are then expressed as constraints on the possible values of the relative positions of the objects involved in that action. As we have mentioned, these constraints are not actually expressed declaratively. Instead, there is a procedure associated with each action which generates the constraints, given a description of the world state.

4. GOAL SATISFACTION

In this section, we will individually discuss the methods that SPAR uses to satisfy operational, geometric and uncertainty-reduction goals. In the course of this discussion, we will frequently allude to the CMS's role in the process of goal satisfaction, however, we will leave a detailed discussion of the CMS for Section 5. For the purposes of this section, it is sufficient to assume that the CMS is capable of determining if a new constraint is consistent with the current constraint set, and if not, signaling failure to the planner.

4.1. Satisfying Operational Goals

In SPAR ensuring the satisfaction of an operational goal proceeds in two steps: finding an action which establishes the goal and then dealing with actions that could clobber (or undo) the goal.

In order to find an action which establishes an operational goal, SPAR first looks at the add lists of the actions which are already in the partially developed plan. If any element of the add list of such an action can be unified with the operational goal, then that unification is performed, and the action is declared to have established the goal. Note that in Chapman's planner, rather than instantiate plan actions using unification, constraints were added to the constraint set indicating that certain symbols were required to "codesignate"*. In SPAR, codesignation constraints are implicit, determined by the unification process rather than noted in the constraint network. Using unification to instantiate actions simplifies the CMS, but costs SPAR some generality (this will be clarified shortly).

If SPAR succeeds in finding an action (already in the current partial plan) whose add list contains an element which can be unified with the goal, that action is constrained to take place prior to the time at which the goal must be satisfied. This is done by adding a prior_to constraint to the constraint network. If the CMS determines that this new ordering constraint is not consistent with the current constraint network, the constraint addition fails and SPAR backtracks

* Codesignation between a variable and constant effectively binds the variable to the constant, while codesignation between two variables indicates that they must be bound to the same constant when instantiated.

in an attempt to find another action in the plan which establishes the goal.

If SPAR fails to find an action in the plan which can establish the goal, it adds one. This consists of instantiating an action template, adding the action to the plan, and constraining the new action to occur prior to the time at which the goal must be satisfied. (This ordering constraint will automatically be consistent with the constraint network, since the action is new and therefore has no other ordering constraints associated with it.) As was discussed in Section 3.1, in order to speed the process of determining which action to add to the plan, and how to properly instantiate that action, SPAR maintains a rule base of actions, indexed by the goals which they establish. This rule base defines the instantiation of plan variables and any initial constraints which are required to satisfy the goal.

Any time SPAR adds an action to the plan, it is possible that the new action may clobber goals which have already been satisfied. For this reason, when a new action is added to the plan, SPAR examines the list of satisfied goals and transfers any of these which could be clobbered by the action to the appropriate pending goal stack. Adding constraints to a plan will never have the effect of clobbering a goal (so long as the new constraint network is consistent). The reason for this is that the constraint network specifies a family of correct plans, any of which will achieve the goals. Adding constraints has the effect of limiting the number of these plans. In other words, adding constraints merely restricts the number of correct plans which SPAR is considering.

Once an operational goal has been established, SPAR examines each action in the current partial plan to see if it could possibly clobber the goal. An action can clobber an operational goal if any element in the action's delete list can be unified with the goal. There are three ways to deal with a potential clobberer. The clobbering action can be constrained to occur after the time at which the goal must be satisfied (promotion of the goal). The goal can be constrained not to unify with the clobbering clause in the action's delete list (separation). An action can be used to re-establish the goal (white knight). This white knight can either be an action which is already in the plan, or it can be a new action which is added specifically for the purpose of re-establishing the clobbered goal.

In SPAR, since pattern matching is done using unification, it difficult to add separation constraints to the plan. The reason for this is that the unification is done using Prolog's unification algorithm, which will not take into account constraints in SPAR's constraint network. Therefore, it is difficult to implement a constraint which says that an element in the delete list of an action, for example holding(part1,Grasp), should not be instantiated so that it matches a particular goal, for example holding(part1,grasp1). For this reason, we have omitted separation as a possible means of declobbering goals in SPAR.

Promotion of the goal is the first option that SPAR tries when declobbering goals. When an action, C, can clobber a goal required to be true during the execution of a certain action, S, SPAR attempts to add a constraint of the form prior_to(S,C), which specifies that the potential clobberer should not be executed until after action S has been executed. If this constraint addition fails, SPAR will attempt to remedy the possible clobbering by the addition of a white knight.

Using a white knight to re-establish a goal is the same as establishing a goal, with the additional condition that the white knight must occur after the potential clobberer. As such, this process proceeds exactly as the establishment process described above, but when a candidate action is found, the additional constraint prior_to(C,W) is added to the constraint network (where C is the action identifier of the clobberer and W is the action identifier of the white knight). In an earlier paragraph we mentioned the possibility of constraining the clobberer to occur before the establishing action. This is the same as allowing the establishing action to act as its own white knight.

4.2. Satisfying Geometric Goals

In SPAR, geometric goals are satisfied by constraining the way in which plan actions are performed. For example, if a geometric goal specifies that the manipulator should be holding an object in a particular grasping configuration, the way to satisfy that goal is to place a constraint on how the manipulator performs the grasping action. In order to do this, SPAR needs to link together the operational and geometric levels of planning. For this purpose, when planning to satisfy operational goals, plan variables are introduced which can be constrained by the geometric level of planning to determine how an action is executed. The geometric preconditions are expressed in terms of those variables. For example, a traditional STRIPS type action is pickup(Object). SPAR's equivalent action is pickup(Object,Grasp). The variable Grasp is used to define the geometric configuration which will be used by the manipulator in grasping the object. At the operational level, the variable Grasp is primarily ignored, but its presence gives SPAR a method of constraining how the pickup operation is actually performed, thus linking distinct levels of planning.

As we pointed out in Section 3.4, geometric goals consists of a set of operational goals and a geometric constraint which is to be applied to the actions that achieve the operational goals. In SPAR, each operational goal that is associated with a geometric precondition of an action is also listed separately as an operational precondition of the action. Therefore, since SPAR only considers geometric goals when the operational goal stack is empty, the operational goals associated with a geometric goal are guaranteed to be satisfied by the current partial plan. Therefore, in order to satisfy a geometric goal, SPAR first finds the actions which establish its associated operational goals, and attempts to constrain the execution of those actions so that the geometric constraint is satisfied. This is done by instructing the CMS to add the geometric constraint to the constraint network. If this succeeds, the goal is satisfied and moved to the list of satisfied goals.

If the CMS determines that the geometric constraint is not consistent with the current constraint network, then one or more new actions must be added to the plan. These new actions are chosen based on the operational goals associated with the geometric goal. The instantiation of the actions' templates proceeds as described in Section 3.1. Once the actions have been added, the appropriate geometric constraint is also added to the constraint network. This constraint will automatically be consistent with the constraint network, since the new action will contain new plan variables which have not yet been constrained. Note that the addition of actions to the plan will introduce new operational goals, and therefore effectively transfer control back to operational planning.

There is no need for SPAR to check for actions that might clobber geometric constraints. The reason for this is that the constraint network has no sense of temporal ordering. The entire network must be consistent at all times. Therefore, if any constraint in the network had the effect of clobbering the new geometric constraint, this would have been detected by the CMS when attempting the constraint addition.

There is also no need to check for actions in the partial plan which might clobber the operational goals associated with a geometric goal. The reason for this is that SPAR only considers geometric goals when the operational goal stack is empty, implying that the operational goals associated with the geometric precondition have been both established and declobbered. A complication arises when an action which achieves an operational goal cannot be constrained to achieve the geometric goal (i.e. the addition of the geometric constraint to the constraint network fails). As we have just described, when this occurs SPAR must add an action to the plan to satisfy the operational goal. This goal satisfaction is done in the same way as described in Section 4.1, and includes declobbering the operational goal.

4.3. Satisfying Uncertainty-Reduction Goals

When there are no remaining operational or geometric goals, SPAR begins the second phase of planning, which deals with uncertainty-reduction goals. There are two fundamental differences between this phase and the first phase of planning. First, the uncertainty-reduction planning does not use the constraint posting method. Second, if no plan instance can be found which satisfies all uncertainty-reduction goals, SPAR does not backtrack to the geometric and operational levels of planning. Instead, it prepares for possible failures by adding verification steps and potential local recovery plans.

As we mentioned earlier, we do not use constraint posting to satisfy uncertainty-reduction goals due to the complexities involved with their representation and evaluation. The high cost of representing uncertainty-reduction goals compared to either operational or geometric goals is partially due to the fact that the geometric and operational effects of actions do not propagate through more than one action, but uncertainties may propagate through many actions. For example, consider the sequence of actions:

action1: pickup(part1,grasp1)
action2: putdown(part1,pos1)
action3: pickup(part1,grasp2)

After the execution of action2, part1 will be in a particular position (which is represented by the variable position1) regardless of where it was prior to the execution of action1. However, the uncertainty in the location of part1 after the execution of action2 will be a function of many variables, including: the uncertainty in the position of the manipulator during the execution of action1 and action2, how the particular grasping configuration used in action1 affects the uncertainty in the location of part1, and the uncertainty in the location of part1 prior to the execution of action1. Therefore, while the geometric preconditions of action3 can be expressed in terms of two plan variables (position1 and grasp2), the uncertainty preconditions depend on every action prior to action3 which involved part1.

The fact that uncertainties can propagate through an indefinite number of actions also affects the complexity of evaluating the uncertainty-reduction constraints. As described in Section 3.2, SPAR uses symbolic algebraic expressions to represent uncertainty. Each time a plan action affects the uncertainty in some quantity, there is, in the worst case, a multiplicative increase in the number of terms in the corresponding symbolic expressions. Therefore, the number of terms in an expression for an uncertain quantity is, in the worst case, exponential in the number of actions in the plan. Since the CMS uses upper and lower bounding routines to evaluate uncertainty-reduction constraints, and since the time complexity of these routines is a function on the number of terms in the input expression, the worst case time complexity for the evaluation of an uncertainty-reduction constraint is exponential in the number of actions in the plan. In contrast, constraints associated with operational and geometric goals can, in the worst case, be evaluated in time that is polynomial in the number of actions in the plan. In the best case, the time is constant (e.g. in evaluating constraints on the robot's joint angles).

There are two reasons for not backtracking into the first phase of planning. First, since SPAR represents uncertainty in the world using bounded sets (e.g. the X location of an object would be represented as $X \pm \Delta X$), even though uncertainty-reduction goals cannot be satisfied, it is quite possible that the actual errors in the world description will be small enough that the plan can be executed without failure. Therefore, SPAR adds verification sensory actions and local recovery plans to offending plan instances, in anticipation of possible execution error. Second, by using the constraint posting approach in the first phase of planning, SPAR attempts to develop the most general plan which will satisfy the operational and geometric goals. Therefore, it is not likely that a great deal could be gained by backtracking into the first phase of planning.

The top level of uncertainty-reduction planning consists of a loop in which specific plan instances are generated and tested until one is found in which all uncertainty-reduction goals can be satisfied. If all possible plan instances have been generated, and none are without violated uncertainty-reduction goals, the instance with the fewest violations is selected for execution.

The uncertainty-reduction planning for a particular plan instance begins with the creation of an augmented plan instance which contains four components: the instantiated list of plan actions (obtained by instantiating the actions from the partial plan that was developed in the first phase of planning so that all constraints in the constraint network are satisfied), an error count (initially set to zero), a list of sensory verification actions (initially set to the empty list), and a list of local error recovery plans (also initially set to the empty list). Once this augmented plan instance has been constructed, SPAR sequentially examines each individual action in the instantiated action list and attempts to satisfy its uncertainty-reduction preconditions. After an action has been considered, its add and delete lists are used to update the world state to reflect the effects of the action. This has the effect of propagating the uncertainty in the world description forward, thereby defining the uncertainty in the world when the next action in the sequence will be executed.

The first step in satisfying an uncertainty-reduction goal for an individual action is the construction of the symbolic algebraic inequality associated with that goal. This is achieved by performing an appropriate combination of matrix multiplications, matrix inversions, etc., as determined by the actual goal. It should be noted that many of the quantities which enter into these operations will be defined in the world state (e.g. the part locations, uncertainties in the part locations). The specific uncertainty-reduction goals for the actions were described in Section 3.2. Again, we note that in SPAR's implementation we have encoded specific procedures to construct the actual expressions. These procedures perform the symbolic matrix operations and also invoke SPAR's algebraic simplifier to reduce the complexity of the resulting expressions.

In SPAR, all of the uncertainty-reduction preconditions are expressed as inequalities with symbolic algebraic expressions on one side of the inequality and a constant value on the other. In order to determine if such an inequality is satisfied, all that is required is to find the upper (or lower) bound on the symbolic expression and see if it is less than (or greater than) the constant value on the opposite side of the inequality. Upper and lower bounds on symbolic expressions are found using the routines SUP and INF, which are described in Section 5.

If the uncertainty in the world description exceeds that which is specified by an uncertainty-reduction goal, SPAR introduces sensing operations into the plan in an attempt to reduce the offending uncertainties. This presents the problem of deciding where in the plan to insert these sensing operations. A general approach would be to go backward through the plan, inserting sensing operations and determining whether or not they reduced the uncertainties to acceptable levels. This type of approach is not necessary in SPAR. Since there are only two types of action which have uncertainty-reduction preconditions, we can predefine the best sensing strategy for each action. For the action pickup(Object,Grasp), we insert a sensing operation immediately prior to its execution to determine a more precise position estimate for Object. For the action assemble(Obj1,Obj2,...), two sensing operations are introduced into the plan instance, just prior to the operation which achieves the goal holding(Obj1,Grasp). These two sensing operations are used to find the more precise estimates of the positions of Obj1 and Obj2.

In SPAR, sensing actions have the same representation as manipulations. For example, the rule to instantiate the template for the "dense_range_scan" action is shown in Fig. 11. Note that

To instantiate the "dense_range_scan(Object)" template:

```
        Generate unique symbols for Theta, Dx, Dy, Dz, ActionId, and Gid

        Add the following information to the database (to identify unc. variables):

                supinf_variable(Dx)
                supinf_variable(Dy)
                supinf_variable(Dz)
                supinf_variable(Theta)

        Add the following bounds on the variables to the database:

                supinf_less(-0.05,Dx)
                supinf_less(Dx,0.05)
                supinf_less(-0.05,Dy)
                supinf_less(Dy,0.05)
                supinf_less(-0.05,Dz)
                supinf_less(Dz,0.05)
                supinf_less(-0.5,Theta)
                supinf_less(Theta,0.5)

        Instantiate the Error Transformation (ErrTm) as:

                tm([    [cos(Theta), sin(Theta), 0],
                        [-1*sin(Theta), cos(Theta), 0],
                        [0,0,1],
                        [Dx,Dy,Dz]]),

        Instantiate the sensing action template as:

                sensing_action(         ActionId,
                                dense_range_scan(Object),
                                preconditions(
                                        [op(Gid,ActionId,gripper(open))],
                                        []),
                                addlist([part_location_unc(Object,ErrTm)]),
                                dellist([part_location_unc(Object,_)])).
```

Figure 11: The rule which instantiates a sensing action template, and adds constraints on the uncertainty variables to the database.

the add and delete lists contain elements which describe how the uncertainties in the world description are reduced by the action. Once the sensing actions have been inserted into the plan instance, these add and delete lists are used to update the world state. The resulting world state is then used to recompute the symbolic expression for the failed uncertainty-reduction goal. Once again, SUP and INF are used to determine if the inequality is satisfied.

If the sensing operations fail to reduce the uncertainty to acceptable levels, SPAR attempts to introduce manipulations into the plan which can reduce the uncertainty. Currently, the only manipulation which is used for this purpose is squeezing an object between the manipulator fingers. The reduction in uncertainty for this action is the same as the reduction in uncertainty for the pickup action, as was described in Section 3.2.3. Since the operational and geometric preconditions for this action are the same as for the pickup action, it can also be spliced into the plan instance just prior to the execution of a pickup action (as with the sensing operations described above), however, the uncertainty in the world description must satisfy the uncertainty-reduction preconditions for the pickup action. For this reason, manipulation to reduce uncertainty is only useful for reducing the uncertainty prior to the assemble action, and then, only for reducing the uncertainty in the location of the object which is resting on the table.

If the sensing operations and manipulations fail to sufficiently reduce uncertainties, SPAR prepares for possible execution error. First, the error count for the augmented plan instance is incremented by one. Second, a sensing verification action and a local recovery plan are added to their respective lists in the augmented plan instance. Figs. 12 and 13 illustrate the various components of the sensing verification action and local recovery plan for for the pickup action. In Fig. 12, the parameters in the parameter list are computed either from the values of plan variables (e.g. the width of the object) or from the results of sensing actions (e.g. the width of the gripper opening). The error-states component consists of a set of condition-action pairs which define the type of error that has occurred. The local recovery plan consists of a set of condition-action pairs that define how to respond to each type of error which may have been encountered.

We should point out that the process of instantiating verification strategies and local recovery plans is in its formative stages. At this point, methods tend to be ad hoc, based on the programmer's evaluation of possible errors and likely recovery plans. We hope that future work will enable us to link CAD modeling systems with SPAR's descriptions of worst case world error to automatically predict the types of errors which could occur, and automatically prescribe verification strategies and recovery plans.

action-id: ActionId,

parameter-list:

> WG = width of object
> W = width of gripper opening

error-states:

> $W = WG \rightarrow$ success
>
> $W = 0 \rightarrow$ error1
>
> true $\rightarrow$ error2

Figure 12: Template for a sensory verification action.

```
action-id:      ActionId,
error-states:
        success → return true
        error1 → local-grope()
        error2 → summon-user()
```

Figure 13: Template for a local recovery plan.

5. CONSTRAINT MANIPULATION

In SPAR, the bulk of the domain knowledge resides in the constraint manipulation system. This allows the top level planning to proceed without any need to "understand" the domain of automated assembly. The actions include preconditions on the geometry of actions and the tolerable uncertainties in the world description, but in order to satisfy these preconditions, the top level planner merely requests that the CMS add constraints to the constraint database. It is the task of the CMS to determine whether or not these new constraints are consistent with the current constraints in the plan, which in turn, requires a certain amount of domain specific knowledge. We could switch the actions which SPAR uses, and the top level planner would not change. However, the CMS would have to be changed to incorporate an understanding of the new types of constraints which it would be forced to deal with.

There are currently three types of constraints which SPAR uses. In operational planning, SPAR uses ordering constraints to ensure that actions are performed in the proper sequence (and that goals are satisfied at the appropriate times). In geometric planning, SPAR uses binary constraints between object positions and manipulator configurations to ensure that the robot will be able to perform the required manipulations. Finally, at the uncertainty-reduction level, symbolic algebraic inequalities are used to express the maximum uncertainty which can exist in the world description prior to the execution of an action.

Throughout the previous sections of the paper, we referred to the CMS maintaining a constraint network. In actuality, there is not a single, uniform constraint network. A directed graph is used for ordering constraints, a binary constraint network is used for the geometric constraints, and algebraic inequalities (expressed in terms of bounded symbolic variables) are used for the uncertainty-reduction constraints. This separation does not interfere with determining the consistency of the constraint set, since the three types of constraints do not interact. For example, even though operational planning might influence the choice of which geometric constraint to add in the course of satisfying a particular geometric goal, once that geometric constraint is chosen, it will be expressed solely in terms of geometric quantities. Therefore, in the constraint database, there will be no interaction between distinct types of constraints.

In this section, we will describe the constraints that are used in SPAR, their semantics, and how the CMS determines whether or not new constraints are consistent with the current constraint set. At this point in time, SPAR's CMS is not complete, in the sense that it is possible that a new constraint will be determined to be inconsistent when it really isn't. The reason for this is that the quantities which enter into the constraints in SPAR are very complex, and often, exact solutions are only approximated. For example, characterizing the space of reachable grasps for a robot entails partitioning a six-dimensional space into reachable and unreachable regions. In SPAR, we have devised a representation of grasping configurations which approximates the true situation. This simplifies the process of constraint manipulation, but adds the possibility that SPAR might miss subtle solutions.

5.1. Ordering Constraints in Operational Planning

In the first phase of planning (used to satisfy operational and geometric goals) SPAR operates as a nonlinear planner. In nonlinear planners, there is not a total ordering of the actions in the plan. Instead, the time of an action's execution is specified by a set of ordering constraints. Each such constraint specifies whether the action should be executed before or after some other action in the plan. While it is possible that the set of ordering constraints in a plan will define a total ordering of the plan steps, more often it will only define a partial ordering.

In SPAR, the only ordering constraint is the prior_to constraint, which is expressed as:

prior_to(Action1, Action2)

and specifies that Action1 must be executed prior to the execution of Action2. When such a constraint is added to the constraint database, the variables Action1 and Action2 will have already been instantiated to the action identifiers for their respective actions.

SPAR's CMS uses a directed graph (which we will refer to as the ordering graph) to keep track of ordering constraints. All actions in the plan are represented in the ordering graph. Any time a new action is added to the plan, a new node is created in the ordering graph, with the action's action identifier as the node's label. An ordering constraint of the form prior_to(Action1,Action2) is represented by an arc directed from the node for Action1 to the node for Action2. Consistency of the ordering constraints is guaranteed as long as the ordering graph contains no cycles, since the only type of inconsistency which might arise is if an action is constrained to occur both prior to, and also after some other action in the plan. This situation is illustrated in Fig. 14.

In order to determine if a new ordering constraint is consistent with the current set of ordering constraints, the CMS checks to see that the new constraint will not result in a cycle in the ordering graph. For a constraint of the form prior_to(Action1,Action2), this is done by searching for a path in the graph from the node for Action2 to the node for Action1. If such a path exists, then the introduction of an arc from Action1 to Action2 will create a cycle, so the CMS returns failure. If no such path exists, the new ordering constraint is added, and the CMS returns success.

5.2. Constraints at the Geometric Level of Planning

All of the geometric constraints in SPAR are either binary constraints between plan variables representing object positions and manipulator positions, or unary constraints on plan variables. Furthermore, both object poses (i.e. possible orientations of objects, not including displacement information) and grasping configurations have been quantized, and assigned labels, so that each of these can be represented by a single, symbolic variable rather than a continuous variable in six dimensional space. Because of these qualities, it is straightforward to represent

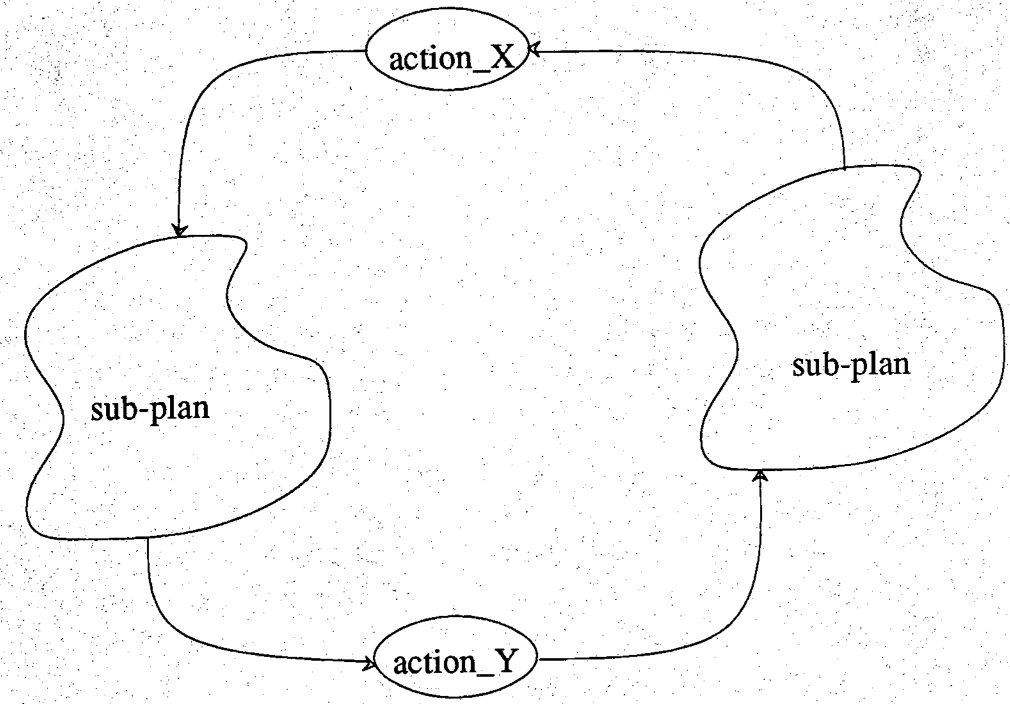

Figure 14: An illustration of an inconsistent ordering graph.

the geometric constraints using a binary constraint network. By using a binary constraint network, when the CMS is instructed to add a new constraint, the consistency of that constraint with the current set of constraints can be determined by adding an arc to the constraint network and then checking the new network for consistency.

We will begin this section with an introduction to binary constraint networks, and an explanation of how such a network is used to represent SPAR's geometric constraints. Following this, we individually describe each type of geometric constraint included in SPAR, and the mechanisms used to evaluate those constraints.

5.2.1. The Geometric Binary Constraint Network

Although this section includes a cursory introduction to constraint networks, those totally unfamiliar with this topic might want to investigate either of [8,9]. We begin our discussion with the following definitions.

Def: The label set for a plan variable is the set of possible values which may be assigned to that variable.

Def: A unary constraint on a variable is a restriction of that variable's label set.

Def: A binary constraint on two variables, V_i and V_j, is a relation

$$C_{ij} \subset L_i \times L_j$$

where L_i is the label set of V_i and L_j is the label set of V_j.

Def: A binary constraint network is an undirected graph whose nodes represent constrained variables, and whose arcs represent constraints between variables.

An example of a binary constraint network is shown in Fig. 15. In this figure, the label set for V1 is {1,2,3}, the label set for V2 is {2,3,4}, and the label set for V3 is {4,5,6}. The constraints are:

$$\text{C12: } V1 + V2 < 5,$$
$$\text{C23: } V2 + V3 < 7,$$
$$\text{C13: } V1 + V3 < 7,$$

A binary constraint network is consistent if there is some instantiation of the variables which simultaneously satisfies all of the constraints in the network. In the example of Fig. 15, there are two consistent instantiations: V1 = 1, V2 = 2, V3 = 4 and V1 = 2, V2 = 2, V3 = 4

SPAR's CMS uses depth first search with backtracking to determine network consistency. For each level in the search, this consists of selecting one node in the network which has not yet been assigned a value, and assigning to it a value which is consistent with all assignments that have previously been made in the search (note that for the first node, there will have been no previous assignments, and so any value from the node's label set may be chosen). The algorithm is illustrated in Fig. 16 and its Prolog implementation is shown in Fig. 17. Note that although the algorithm in Fig. 16 generates all consistent labels for a node prior to investigating any single

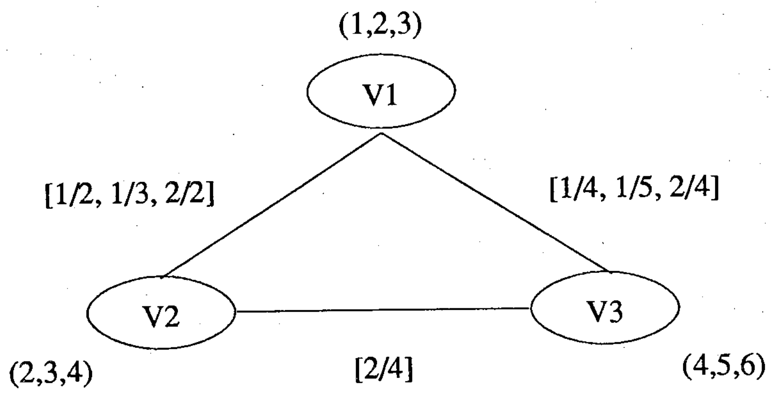

Figure 15: Example binary constraint network.

```
csp(Nodes, Constraints, PrevAssns)

    if (Nodes = $\varnothing$) return PrevAssns

    Node $\leftarrow$ head of Nodes
    Rest $\leftarrow$ tail of Nodes

    Labels $\leftarrow$ find_consistent_labels(Node,PrevAssns,Constraints)

    foreach Label $\in$ Labels
    {
        NewAssns $\leftarrow$ concat(Node/Label,PrevAssns)
        Result $\leftarrow$ csp(Rest,Constraints,NewAssns)

        if (Result $\neq$ nil) return NewAssns
    }

    return nil
```

Figure 16: Recursive algorithm to check for network consistency.

```
csp(VarSet,Constraints,Result) :-
    csp(VarSet,Constraints,[],Result).

csp([],_,R,R).
csp([V|Rest],Constraints,Asns,R) :-
    choose_one(V,Constraints,Asns,Label),
    csp(Rest,Constraints,[V/Label|Asns],R).

choose_one(Var,Constraints,PrevAssns,Label) :-
    get_label_set(Var,LabelSet,Constraints),
    member(Label,LabelSet),
    all_consistent(Var/Label,PrevAssns,Constraints).

all_consistent(_,[],_).

all_consistent(Var/Label,[V1/L1|Rest],Constraints) :-
    get_constraint_pairs(Var,V1,Pairs,Constraints),
    !,
    member(Label/L1,Pairs),
    all_consistent(Var/Label,Rest,Constraints).

all_consistent(Var/Label,[_|Rest],Constraints) :-
    all_consistent(Var/Label,Rest,Constraints).
```

Figure 17: Prolog implementation of network consistency algorithm.

assignment, the Prolog implementation uses backtracking to successively generate candidate labels.

In order to represent SPAR's geometric constraints using a binary constraint network, each geometric plan variable (e.g. grasp configurations, positions) is represented by a node in the network. When a new variable is introduced into the plan, a node is added to the network and assigned an initial label set. This label set is merely the set of values which may be assigned to that variable (determined by the action template instantiation rules discussed in Section 3.1). For example, if the variable represents a grasping configuration for a particular object, then the initial label set for its node in the constraint network will contain the labels of all of the grasping configurations for that object (grasping configurations will be described in Section 5.2.4).

Binary constraints between plan variables are represented by arcs between the corresponding nodes in the network (these arcs are not directed). Each arc in the network contains a set of pairs of values which indicate the valid pairs of labels for the connected nodes. Determining the valid pairs of labels requires a semantic understanding of the domain, but once the pairs have been assigned, no domain knowledge is required to check for network consistency. Therefore, we can apply the consistency algorithm shown in Figs. 16 and 17.

When the CMS is instructed to add a unary constraint to the network, it first updates the label set of the appropriate node, and then updates each arc connected to that node by deleting pairs which are no longer valid given the node's new label set. Finally, the new network is checked for consistency. When the CMS is instructed to add a new binary constraint to the network, it adds the appropriate arc between the appropriate nodes (creating the nodes if they do not already exist in the network), and then checks for network consistency.

5.2.2. Set Membership

In order to restrict the label set of a plan variable, SPAR uses the constraint member(Variable,Labels). If there is no node in the geometric constraint network for Variable, the CMS adds one, and assigns its initial label set to contain the elements of Labels. If there is already a node in the constraint network for Variable, the CMS takes two steps to ensure that the new constraint on Variable's label set will not result in an inconsistent network. The first step ensures node consistency (i.e. that the node for Variable will have at least one possible label), and the second ensures network consistency. To ensure node consistency, the set Labels is intersected with Variable's current label set. If the intersection is empty, then the new member constraint is not consistent with the current constraint set. If the intersection is not empty, then it is assigned as Variable's new label set. To ensure network consistency, all arcs leaving the node corresponding to Variable are updated by deleting pairs which assign Variable a value which is not in its new label set. The new network is then checked for consistency using the algorithm shown in Figs. 16 and 17. If both node and network consistency are satisfied, the CMS returns success. If not, failure is returned.

5.2.3. Stable Poses and Position Classes

For the purpose of assembly operations, the exact position of an object is not always important. What is important is that the object be oriented in a way that allows the mating features of the object to be accessible. For example, if the assembly operation is to insert a peg into a hole in a block, it is not important how the block is oriented, as long as the hole is positioned so that the peg can be inserted. In light of this, in SPAR, we characterize object positions using equivalence classes. These classes are based on the object's stable poses, where by stable pose we mean an orientation of the object which allows it to rest naturally on the work table. For example, a cube has six stable poses.

The use of stable poses to quantize the space of object positions serves two purposes. First, it provides a method for easily determining which of an object's features will be obscured by the work table. Second, when the plan calls for an object to be placed in some position (by the putdown action), most often the displacement of the object is not important. Stable poses provide a method of specifying destination positions in terms of the object's orientation, without regard to the actual X,Y,Z position. Clearly, in a cluttered work cell, objects will not always be found in one of their stable poses. However, since stable poses are only used to determine a list of occluded features and to specify destinations of held objects, this will not be a problem as long as the sensory system is capable of determining by inspection the object's occluded features.

Fig. 18 illustrates the data structure used to represent a single stable pose of an object. The structure includes the name of the object, the label for the particular stable pose, a list of faces on which the object is resting (the rfaces), a list of faces which are occluded in the pose (the ofaces), and a homogeneous transformation which specifies the orientation of the object for the pose. Note that rfaces and ofaces need not be equal. It is often the case that the rfaces is a subset of ofaces. Also note that the transformation used to represent the pose is not unique. Since the stable pose will only fix two rotational degrees of freedom, any rotation about the world Z-axis would result in the same stable pose.

Using this representation for object positions, geometric goals about object locations can be expressed in terms of set membership. That is, the planner can determine the set of stable poses which are allowable for a certain assembly operation and constrain the object's position to correspond to one of those poses. For this purpose, SPAR uses the constraint in_position_class(Position,Plist). This constraint indicates that the orientation specified by Position must correspond to one of the stable poses in Plist.

If Position is instantiated to a homogeneous transformation which represents both the orientation and displacement of an object (for instance if the position of the object has been ascertained by the sensing system), then this constraint cannot be evaluated by a simple membership test. In this case, the CMS must determine to which stable pose of the object Position corresponds. This can be done in one of two ways. If the object is resting on the table, it is a simple matter to compare the rotational component of Position to the rotations specified by

```
pose(           part1,
                part1_p2,
                rfaces([5]),
                ofaces([5,6]),
                tm([[0,0,1],[1,0,0],[0,1,0],[0,0,0]]))
```

Figure 18: Data structure for stable pose.

the various stable poses of the object to determine in which stable pose the object is resting. If the object is not resting on the table (e.g. if it is leaning against some other object in the work cell), then the sensing system must be used to determine the set of object features which are occluded. Position is then determined to correspond to the stable pose which obscures the same set of features.

Aside from the situation described in the last paragraph, the CMS handles the addition of an in_position_class constraint in the same way that it handles the member constraint. It restricts Position's label set, updates the arcs which are connected to Position's node, and then checks for network consistency.

5.2.4. Reachability of Grasps

Whenever the planner inserts a manipulation action into the plan, it must ensure that all of the configurations required to perform that manipulation will be physically realizable. In order to do this, SPAR uses two constraints:

reachable(Grasp,Position)

and

mate_reachable(Grasp,Position,T_a)

The first of these indicates that if the object to be manipulated is in the position specified by the variable Position, and the configuration used to grasp the object is specified by Grasp, then that combination must be physically realizable. This constraint is used both in grasping, and in placing objects. The second constraint is used for mating operations, where T_a is a homogeneous transformation which represents the destination position of the grasped object relative to the coordinate frame specified by Position.

For specific values of Grasp and Position, two conditions must be met in order for the reachable constraint to be satisfied.

1: The faces of the grasped object which come into contact with the manipulator fingers must not be in contact with the table (or any other object) when the object is located in Position.

2: The robot must be able to perform the grasp without exceeding any of its physical joint limits.

For the mate_reachable condition, only the second condition is used. However, the position which the manipulator must reach is not Position, as in the reachable constraint, but $T_O T_a$, where T_O is the homogeneous transformation corresponding to Position.

To verify condition 1, the system must invoke the object modeling system to determine which features of the object will be in contact with the table when the object is in Position, and which features of the object will be in contact with the manipulator when the object is grasped in the configuration specified by Grasp. (We should note that the modeling system used in SPAR is not a CSG modeler. A number of object representations are included in an object model, including a grasping model, a table of the stable poses, and a great deal of geometric information which is used by the sensing system for object recognition and localization.) If Position corresponds to one of the object's stable poses, a simple table lookup operation is used to determine which features are in contact with the table. If Position is an absolute position, then the system must determine the set of occluded features as was discussed in Section 5.2.3.

Condition 2 is verified, for specific values of Grasp and Position, by invoking routines which compute the inverse kinematic solution for the robot's joint angles given an absolute position of the end effector. This is done as follows. A particular grasp has associated with it a homogeneous transformation which defines the coordinate frame of the robot manipulator relative to the frame of the object. We will refer to this as the grasp transformation, or alternatively T_g. If Position is an absolute position (i.e. it has a specific X,Y,Z location as well as a specified orientation) specified by the homogeneous transformation T_O, we compute T, the transformation representing the manipulator's coordinate frame relative to the world frame, by $T = T_O T_g$. In the mate_reachable case, $T = T_O T_a T_g$. This transformation is used as the input to the inverse kinematics program. The joint angles which are found by this program are then tested to ensure that they are within the limits attainable by the robot. Currently, our lab is using a PUMA 762 robot for manipulation experiments. Descriptions of the kinematic and inverse kinematic solutions for this type of robot can be found in [21].

If Position corresponds to a stable pose (that is, it specifies an orientation of the object, but no absolute X,Y,Z position) the CMS assumes that condition 2 can be satisfied by some suitable choice of X,Y,Z. That is, we assume that for any arbitrary orientation of the robot manipulator, there will be some location in the work space where this orientation can be physically performed (where by orientation, we mean that the coordinate frame for the grasp has axes whose origin is not specified, but whose orientation relative to the world frame is specified).

When the CMS is instructed to add either a reachable or mate_reachable constraint to the constraint network, the two conditions described above are used to determine all valid pairs of values for Grasp and Position (note that T_a will always be instantiated to a constant homogeneous transformation). This is done by exhaustively pairing every value from the label set for Grasp with every value from the label set for Position, and recording all pairs which satisfy the two conditions. These pairs are then used to construct a new arc connecting the nodes for Grasp and Position. Finally, a network consistency check is performed. If the consistency check fails, the CMS signals failure and the old network is restored. Otherwise, the CMS signals success and retains the new network.

Exhaustive enumeration of pairs of positions and grasps is not as difficult as it might seem. First, as we have described earlier, there are a finite number of possible stable poses associated with any object (if the object is in a known location determined by the sensing system, then there is only one position to consider). Usually this number is fairly small. Second, we quantize the space of grasping operations based on the features of the object which are obscured by the grasp, and the features of the object which come into contact with the manipulator fingers in the grasp. This approach is similar to that described in [29,41].

In the object model, a grasp configuration is specified by a set of constraints on the X,Y,Z location of the manipulator's tool center relative to the origin of the object's coordinate frame, and a set of constraints on the Euler angles (ϕ, θ and ψ) that specify the orientation of the manipulator relative to the object frame. Currently, in order to determine if condition 2 above is satisfied for a particular grasping configuration, the system uses a "characteristic" T_g. This T_g is derived by assigning specific values to each of X,Y,Z, ϕ,θ and ψ, and deriving the corresponding homogeneous transformation. An alternative to this approach would be to check the joint angles required to perform the grasp at the boundary cases for the configuration (i.e. a number of specific T_g's would be created by looking at the boundary cases of the constraints on X,Y,Z and the Euler angles defining the configuration).

Fig. 19 illustrates the data structure used to represent a grasping configuration for an object. This structure includes the name of the object, the label for the particular grasping configuration, a list of faces of the object which contact the manipulator fingers during the grasp (gfaces), a list of faces of the object which are obscured by the grasping configuration (ofaces), the characteristic T_G (expressed as X,Y,Z,ϕ,θ and ψ), and the width of the object in the direction parallel to the manipulator's sliding axis.

By making this type of quantization of the space of grasping configurations, we replace exact descriptions of grasping configurations with approximations. Because of this, it is possible that SPAR will occasionally determine that a reachable constraint is not consistent with the current constraint database, when in fact it is consistent. In general, we do not expect this to happen except when the manipulations which are to be performed require the robot to operate near the boundaries of its work envelope.

5.3. Constraints at the Uncertainty-Reduction Level of Planning

As we described in previous sections, when the planner considers the uncertainty-reduction goals, it does so for a particular plan instance. As a consequence of this, at the time of their evaluation, the uncertainty-reduction goals (which are expressed as symbolic algebraic inequalities) will be expressed in terms of specific bounded symbolic variables. Therefore, determining if an uncertainty-reduction goal is satisfied consists of a single evaluation (rather than a series of evaluations as was required in the geometric constraints).

In general, given an uncertainty-reduction goal, G, which is expressed as an algebraic inequality, and given a set of constraints, C, on the variables contained in G, we need to be able to determine whether or not G is guaranteed to be satisfied given C. One approach to this problem is to examine the satisfying sets for G and C. We represent the satisfying set for a constraint set C by sat(C), and define sat(C) as follows. Given a constraint set C, on the N variables, $v_1 \cdots v_N$, then sat(C) is an N-dimensional space such that $<c_1, c_2, \cdots c_N> \in$ sat(C) if and only if C is satisfied when $v_1 = c_1 \cdots v_N = c_N$.

Given this definition of satisfying sets, it is apparent that the uncertainty-reduction goal G will be satisfied if and only if sat(C) $\subseteq$ sat(G). In other words, a necessary and sufficient

```
grasp(          part1,
                part1_g7,
                gfaces([8,10]),
                ofaces([8,10,7]),
                eulerxyz(8.0,1.75,2.0,0.0,-90.0,0.0),
                width(3.5))
```

Figure 19: Data structure for grasping configuration

condition for satisfaction of an uncertainty-reduction goal for an action is that the space which defines the uncertainty in the current world description is contained in the space which defines the permissible uncertainty for the action.

An alternative approach, and the one which we use in our system, is based on the fact that all uncertainty-reduction goals are expressed as inequalities of the form $\text{expr}_1 < \text{expr}_2$. If we find the maximum value for expr_1 and the minimum value for expr_2, under the constraints contained in C, we can determine whether the uncertainty-reduction goals are met simply be checking to see if $\max(\text{expr}_1) < \min(\text{expr}_2)$. This test is sufficient for showing that G is satisfied by C, but it is not necessary. That is, it is possible that G can be satisfied and $\max(\text{expr}_1) \nless \min(\text{expr}_2)$. We have restricted our uncertainty-reduction goals to have constant values on at least one side of the inequality, so this case does not arise. By taking this approach, we have reduced our problem to finding upper and lower bounds on symbolic expressions, subject to a set of constraints.

At this point we should explain that the set of constraints on the values for the symbolic variables in the uncertainty-reduction goals is maintained in a global storage area. The reason for this is that the bounds on such variables will never change, so when a symbolic variable is introduced, the bounds on that variable are immediately recorded. At first this may seem contradictory to what has been said in previous sections about reducing uncertainty in the world description. However, when the uncertainty in some quantity is reduced (for example by the application of a sensing operation), the expression for the uncertainty in that quantity is updated, so that it is expressed in terms of a new set of bounded variables. By constraining the bounds on these new variables to be tighter than the bounds on the previous variables, a reduction in the uncertainty in the world description is effected.

In order to find upper and lower bounds on symbolic expressions, we have implemented a system very similar to the SUP/INF system which was introduced by Bledsoe [2], and then refined by Shostak [35], and later Brooks for his ACRONYM system [4]. The functions SUP and INF each take two arguments, a symbolic expression and a set of variables, and return upper/lower bounds on the expression in terms of the variables in the variable set. The method SUP/INF employs is to recursively break down expressions into subexpressions, find bounds on these subexpressions, and then combine the bounds using rules from interval arithmetic. Obviously this works for linear expressions where superposition holds. When expressions are nonlinear, however, it is quite possible that the bounds on the individual subexpressions will be looser than the bounds on the subexpressions when considered in the context of the whole expression. Because of this, it is possible that SUP/INF will sometimes find bounds which are not exact.

In spite of this disadvantage, the policy of recursively finding bounds on subexpressions and then combining those bounds guarantees that the algorithms will terminate. This has been shown by Shostak for his version of SUP/INF, and later by Brooks for his modified versions. Furthermore, even though it is possible that SUP/INF will not return exact bounds, it has been shown (again, by Shostak and later by Brooks) that they are conservative, in that SUP always returns a value which is greater than or equal to the maximum, and INF always returns a value less than or equal to the minimum. The fact that SUP/INF sometime only approximates solutions is not a severe problem for SPAR, since failure to satisfy uncertainty constraints has a worst case result of the addition of sensing actions to the plan. That is, if the CMS determines that the uncertainty constraints cannot be satisfied, it does not backtrack. It merely prepares for the possibility of failure.

6. A TASK PLANNING EXAMPLE

In this section, we will illustrate SPAR's flow of control with an assembly example. Consider the assembly task shown in Fig. 20. The assembly goal is to have the peg inserted into the block so that the small hole in the block is aligned with the hole in the peg's base. The user specifies this with a goal of the form:

assembled(peg,block,Msurfaces,Tm,Va)

where Msurfaces is instantiated to a two element list, the first element being a list of the peg's surfaces which will come into contact with the block, and the second element being a list of the block's surfaces which will come into contact with the peg. The variable Tm is instantiated to a homogeneous transformation matrix which represents the goal position of the peg relative to the position of the block. The variable Va is instantiated to a vector which specifies the approach for the mating operation relative to the position of the block. In other words, the user specifies the positions of the parts relative to one another in the goal configuration, as well as the relative locations prior to the goal.

In order to satisfy this goal, SPAR examines its possible actions, and selects the assemble action shown in Fig. 4. Of course, the assemble action has both operational and geometric preconditions which must now be considered, so the planner pushes these onto the appropriate goal stacks. The goal stacks and plan action list are shown in Fig. 21.

At this point, a word about the meaning of the preconditions is in order. Note that in the assemble action there is a precondition of the form:

```
geo(GoalId1,ActionID,
    in_position_class(Position,PositionList),
    part_location(Obj2,Position))
```

As we discussed in Section 5, SPAR associates a set of stable poses with each object, where, by stable pose, we mean an orientation in which the object will rest naturally on the table. In order to mate two objects, SPAR requires that the stationary object be in one of its stable poses which does not obscure any of its mating features. The set of stable poses which satisfy this condition is easily determined by comparing the value of each stable pose's ofaces set with the set of

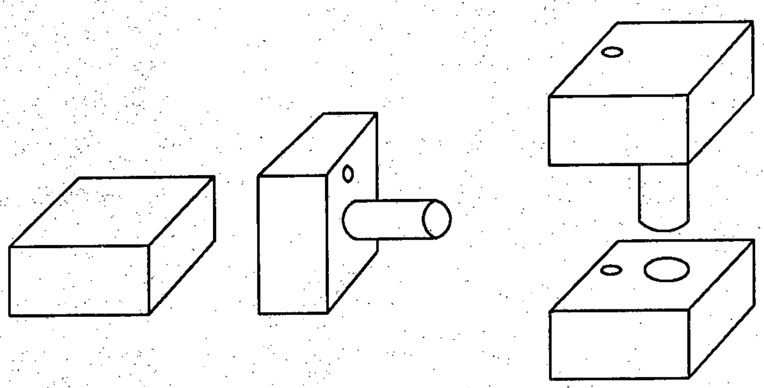

Figure 20: The initial state, and assembly goal for the example.

OPERATIONAL GOAL STACK

```
op(goal_1, action_1, holding(peg, Grasp))

op(goal_2, action_1, part_location(block, Pos_1))
```

GEOMETRIC GOAL STACK

```
geo(goal_3, action_1,
        member(Grasp, [p1G1, ... p1Gn]),
        holding(peg, Grasp))

geo(goal_4, action_1,
        in_position_class(Pos_1, [p2P1, p2P2, ... p2P6]),
        part_location(block, Pos_1))

geo(goal_5, action_1,
        mate_reachable(Grasp, Pos_1, trans_1),
        part_location(block, Pos_1)
        holding(peg, Grasp))
```

PLAN ACTIONS

```
action(action_1,assemble(
                        peg,
                        block,
                        [[peg_face], [block_face]],
                        trans_1, vec_1))
```

Figure 21: Goal stacks and plan actions after the addition of the "assemble" action.

mating features. When the planner adds the assemble action to the plan, it instantiates the variable PositionList to this list. (We should note that the list of stable poses is actually a list of pointers to the data structures for the stable poses.)

This same kind of instantiation takes place for the precondition

```
geo(GoalId2,ActionId,
        member(Grasp,GraspList),
        holding(Obj1,Grasp))
```

In our system, grasping configurations specify not only the geometric configuration which is used to grasp the object, but also the set of object features which are obscured by the grasp (as was discussed in Section 5.2.4). Therefore, it is a simple matter to determine which grasping configurations do not obscure the mating features of the object. When the planner adds the assemble action to the plan, it instantiates the variable GraspList to be this set of grasping configurations.

Fig. 21 shows the instantiated versions of the preconditions, as they appear on the goal stacks. Note that the variables used to identify the preconditions and the action to which the preconditions correspond have also been instantiated.

The first operational goal is that the gripper be holding the peg in some valid grasp (remember that at the operational level, SPAR is not concerned with the grasp beyond this condition). Since it is not possible to merely add a constraint to the plan to achieve this goal (i.e. there is no existing action in the plan whose execution can be constrained so that it results in the manipulator holding the peg), SPAR inserts the action pickup(peg,grasp_1) into the plan, with the constraint that the pickup action must occur prior to the mating action. This results in the addition of an arc to the ordering graph, directed from action2 to action1. The preconditions of the pickup action are then pushed onto the appropriate goal stacks. The resulting goal stacks are shown in Fig. 22. Note that when the planner adds this action, it instantiates the variable Grasp to the label grasp_1, and that this instantiation affects all appearances of Grasp on the goal stacks.

The remaining operational goals are trivially satisfied by the initial world state, so the planner moves them to the satisfied goal list and turns to its geometric goals. (Note that when the goals are satisfied, instances of the variable Pos_1 and Pos_2 on the goal stack are instantiated to init_pos1 and init_pos2. The corresponding label sets are constrained to contain single elements which are the homogeneous transformations representing the block's and peg's initial positions.) The top goal on the geometric goal stack, goal_8, is for the pickup action, and it specifies that the manipulator configuration used to pickup the peg, grasp_1, be physically realizable by the robot. To do this, the planner attempts to add a constraint on the way in which grasp_1 is chosen, so that the configuration will be reachable. This is done by instructing the CMS to add the constraint reachable(grasp_1,init_pos2) to the constraint network. For our example, let us consider that this constraint is consistent with the constraint network.

OPERATIONAL GOAL STACK

```
op(goal_6, action_2, gripper(open))

op(goal_7, action_2, part_location(peg, Pos_2))

op(goal_2, action_1, part_location(block, Pos_1))
```

GEOMETRIC GOAL STACK

```
geo(goal_8, action_2,
        reachable(grasp_1, Pos_2),),
        part_location(peg, Pos_2))

geo(goal_3, action_1,
        member(grasp_1, [p1G1, ... p1Gn]),
        holding(peg, grasp_1))

geo(goal_4, action_1,
        in_position_class(Pos_1, [p2P1, p2P2, ... p2P6]),
        part_location(block, Pos_1))

geo(goal_5, action_1,
        mate_reachable(grasp_1, Pos_1, trans_1),
        part_location(block, Pos_1)
        holding(peg, grasp_1))
```

PLAN ACTIONS

```
action(action_1,assemble(
                        peg,
                        block,
                        [[peg_face], [block_face]],
                        trans_1, vec_1))

action(action_2,pickup(peg,grasp_1))
```

Figure 22: Goal stacks and plan actions after the addition of the "pickup" action.

The next goal on the geometric goal stack, goal_3, specifies that grasp_1 must not obscure any of the mating features of the peg. This is expressed as a member constraint, that is, a restriction on the label set for the plan variable grasp_1. Again, the planner invokes the CMS to add the member constraint to the constraint network. Again, for the example, let us suppose that this succeeds. Note that if adding this constraint resulted in an inconsistent constraint set SPAR would be forced to insert additional manipulations.

Up to this point in the example, SPAR has been able to satisfy geometric goals merely by adding constraints on the way in which operations are performed. In some cases, it will not be possible to satisfy geometric goals this way, and an alternative approach must be used. This is the case for the geometric precondition goal_4, which constrains the possible positions of the block. Consider the situation when the block is face down in the initial world state, as shown in Fig. 20. Since there is no action currently in the plan which manipulates the block, SPAR cannot constrain the execution of a plan action to achieve the goal. Furthermore, the planner cannot add a constraint on the block's initial position, because it is a constant value which is defined by the initial world state. Therefore, backtracking must be used to find some alternative method to satisfy the goal goal_2, which specifies the position of the block.

Remember that goal_2 was originally satisfied by the initial world state. On backtracking, SPAR will try to find some other action in the plan to satisfy goal_2. As mentioned above, there is no action in the current plan which can accomplish this. Therefore, SPAR adds the action putdown(block,pos_1) to the plan. When SPAR reconsiders goal_4, the value of pos_1 will be constrained so that no mating features of the block are in contact with the table when the block is in this position. This amounts to constraining pos_1 to be any of the block's stable poses other than p2P3, the single configuration which obscures the hole. In addition, SPAR adds the constraint prior_to(action_3,action_1) to the ordering graph, since the block must be put down prior to the assemble action. Of course the addition of this plan action introduces new goals, and so additional planning must be done. This planning, however, is very similar to the planning which must be done to pick up the peg appropriately, and so we will not discuss it here.

The final result of the first phase of planning is shown in Figs. 23-25. Fig. 23 shows the four actions which are required. The top of Fig. 24 shows the geometric binary constraint network, which can be interpreted as follows. The grasping configuration grasp_2 is used to pick up the block, and then to place it on the table. Therefore, both init_pos_1 and pos_1 must be reachable using grasp_2. This is indicated by the arcs connecting grasp_2 to init_pos_1 and pos_1. Similarly, grasp_1 is used to pick up the peg, and then to assemble the peg to the block (which is now located in pos_1). The possible pairs of values for each of these arcs are shown in Fig. 25, as are the label sets for the nodes. The possible pairs of values for each arc in the network are determined by examining each possible pair of values from the label sets of the connected nodes, and collecting those which meet the conditions outlined in Section 5.2.4. In the figure, we have represented stable poses by symbols of the form pxPy, where x is used to indicate the object (the peg is indicated by x=1, the block by x=2) and y is used to indicate the

Plan Actions

```
        action(action_4, pickup(block, grasp_2))

        action(action_3, putdown(block, pos_1))

        action(action_2, pickup(peg, grasp_1))

        action(action_1, assemble(
                                  peg,
                                  block,
                                  [[peg_face], [block_face]],
                                  trans_1,
                                  vec_1))
```

Figure 23: Plan actions solve the assembly task shown in Fig. 14.

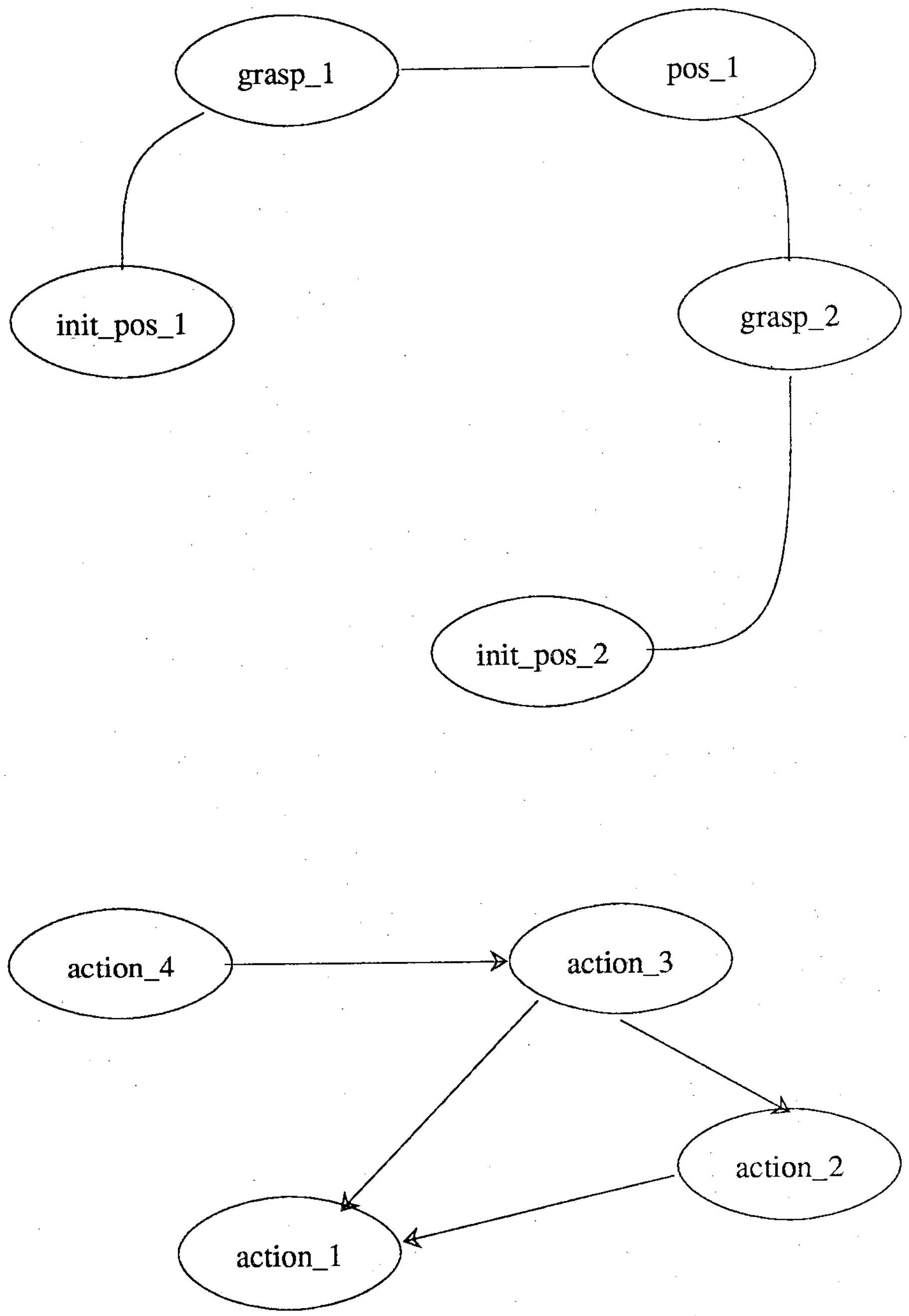

Figure 24: Constraint network for the assembly plan to solve the task shown in Fig. 20.

NETWORK ARCS

grasp_1-pos_1:

[p1G1/p2P1,p1G1/p2P2,p1G1/p2P4,p1G1/p2P5,p1G1/p2P6,p1G11/p2P1,p1G11/p2P2,
p1G11/p2P4,p1G11/p2P5,p1G11/p2P6,p1G12/p2P1,p1G12/p2P2,p1G12/p2P4,
p1G12/p2P5,p1G12/p2P6,p1G13/p2P1,p1G13/p2P2,p1G13/p2P4,p1G13/p2P5,
p1G13/p2P6,p1G14/p2P1,p1G14/p2P2,p1G14/p2P4,p1G14/p2P5,p1G14/p2P6,
p1G15/p2P1,p1G15/p2P2,p1G15/p2P4,p1G15/p2P5,p1G15/p2P6,p1G17/p2P1,
p1G17/p2P2,p1G17/p2P4,p1G17/p2P5,p1G17/p2P6,p1G18/p2P1,p1G18/p2P2,
p1G18/p2P4,p1G18/p2P5,p1G18/p2P6,p1G20/p2P1,p1G20/p2P2,p1G20/p2P4,
p1G20/p2P5,p1G20/p2P6,p1G3/p2P1,p1G3/p2P2,p1G3/p2P4,p1G3/p2P5,
p1G3/p2P6,p1G6/p2P1,p1G6/p2P2,p1G6/p2P4,p1G6/p2P5,p1G6/p2P6,p1G7/p2P1,
p1G7/p2P2,p1G7/p2P4,p1G7/p2P5,p1G7/p2P6,p1G8/p2P1,p1G8/p2P2,p1G8/p2P4,
p1G8/p2P5,p1G8/p2P6,p1G9/p2P1,p1G9/p2P2,p1G9/p2P4,p1G9/p2P5,p1G9/p2P6]

grasp_2-pos_1:

[p2G1/p2P1,p2G1/p2P4,p2G10/p2P2,p2G10/p2P4,p2G11/p2P2,p2G11/p2P6,
p2G12/p2P2,p2G12/p2P5,p2G13/p2P1,p2G13/p2P5,p2G13/p2P6,p2G14/p2P1,
p2G14/p2P6,p2G15/p2P1,p2G15/p2P5,p2G16/p2P5,p2G2/p2P1,p2G3/p2P4,
p2G4/p2P5,p2G4/p2P6,p2G5/p2P5,p2G5/p2P6,p2G6/p2P5,p2G6/p2P6,
p2G7/p2P6,p2G8/p2P1,p2G8/p2P2,p2G8/p2P4,p2G9/p2P1,p2G9/p2P2])

grasp_2-init_pos2:

[p2G1/tr2,p2G16/tr2,p2G2/tr2,p2G3/tr2,p2G4/tr2,p2G5/tr2,p2G6/tr2,p2G7/tr2]

grasp_1-init_pos1:

[p1G12/tr1,p1G3/tr1,p1G7/tr1,p1G8/tr1]

LABEL SETS

pos_1: [p2P1,p2P2,p2P4,p2P5,p2P6]

init_pos1: [tr1]

init_pos2: [tr2]

grasp_2: [p2G1,p2G2,p2G3,p2G4,p2G5,p2G6,p2G7,p2G8,p2G9,
p2G10,p2G11,p2G12,p2G13,p2G14,p2G15,p2G16]

grasp_1: [p1G1,p1G3,p1G6,p1G7,p1G8,p1G9,p1G11,p1G12,p1G13,
p1G14,p1G15,p1G17,p1G18,p1G20]

Figure 25: The arcs and label sets for the constraint network shown in Fig. 24.

specific stable pose for the object. Symbols representing grasping configurations have a similar interpretation.

The bottom of Fig. 24 shows the ordering graph. Note that in the ordering graph, in addition to the arcs we have mentioned above, there is an arc from action_3 to action_2. This arc is added to the graph because the pickup action used to pick up the block (action_4) clobbers the gripper(open) operational goal for the pickup action used to pick up the peg (action_2). To remedy this, the putdown action (action_3) is constrained to come between action_4 and action_2, to act as a white knight, and reestablish the gripper(open) goal.

Once the operational and geometric goals have been satisfied, SPAR considers the uncertainty-reduction goals. As we have described earlier, SPAR chooses a specific instance of the plan (which satisfies the constraint network), and propagates uncertainties forward through the plan actions to determine if the uncertainty-reduction goals are satisfied. For this example, we will only consider the uncertainty-reduction goals for the first pickup action, which were discussed in Section 3.2.

In order to evaluate the constraints associated with these goals, SPAR invokes the procedure which constructs and evaluates the symbolic constraint for the goal. This involves symbolic matrix multiplication, symbolic matrix inversion and symbolic algebraic simplification. The resulting expression for Y component of $P_1^{-1} C_1$:

```
-2.0 + 3.0*cos(thetagr)*cos(theta_o) + cos(thetagr)*dx_o + sin(thetagr) +
sin(thetagr)*dxgr + -1.0*cos(thetagr)*sin(theta_o) + -1.5*cos(thetagr) +
-3.0*sin(thetagr)*sin(theta_o) + -1.0*cos(theta_o)*sin(thetagr) +
-1.0*sin(thetagr)*dy_o + -1*cos(thetagr)*dygr
```

Note that thetagr, dxgr, dygr and dzgr represent the uncertainties in the gripper configuration, and theta_o, dx_o, dy_o and dz_o represent the uncertainties in the object position. Also, for this particular plan instance, W_p was three inches and W_m was four inches. The complexity of this expression illustrates the reasons we outlined in Section 4.3 for applying uncertainty-reduction planning to specific plan instances instead of using a constraint posting approach.

Similar expressions are found for the remaining terms, but we will omit these here. Using the SUP and INF routines given the bounds on the uncertainties listed in Table 2, the lower bound on this expression is found to be 3.2793, which indicates that the constraint was satisfied. The remaining three constraints are evaluated in a similar fashion.

If the uncertainty-reduction goals are not satisfied in the world description, SPAR attempts to add a sensing operation to the plan. Sensing operations are represented in the same way as manipulations. In particular, they have a set of preconditions which must be met before they are applied (e.g. to perform a range scan, the manipulator must be free) and an add/delete list which specifies the sensing actions affect on the world description. The "dense_range_scan" sensing action is shown in Fig. 11. Obviously it is impossible to predict the results of a sensing action, so the add/delete lists merely characterize the possible reduction in uncertainty for the sensing

Table 2: Bounds on uncertainty variables used in the example of Section 4.5.

Bounds on Uncertainty Variables		
Variable	Lower Bound	Upper Bound
dxgr	-0.001	0.001
dygr	-0.001	0.001
dzgr	-0.001	0.001
thetagr	-0.001	0.001
dx_o	-0.11	0.11
dy_o	-0.11	0.11
dz_o	-0.11	0.11
theta_o	-5.5	5.5

operation by describing the uncertainty in the object's location after the application of the sensing operation. For example, the part_location_unc fact in the add list of the range-scan action indicates that dense_range_scan-ning an object will reduce the uncertainty in the X,Y and Z locations of the part to an amount less than 0.05 inches, and the uncertainty in the rotation about the world Z axis to an amount less than 0.5 degrees. If this reduction is sufficient (this is tested by again evaluating the uncertainty-reduction constraints, but with the dense_range_scan's description of the object location uncertainty) the dense_range_scan operation is inserted into the plan.

If SPAR cannot sufficiently reduce the uncertainty in the peg's location, it augments the plan instance with verification sensing operations and local recovery plans. For the pickup action, these are as shown in Figs. 12 and 13.

7. DISCUSSION

In this section, we discuss a number of issues which are helpful in evaluating SPAR.

7.1. Correctness and Completeness

Two important issues in planning systems are correctness and completeness. By correctness, we mean that goals are satisfied when the planner says that they are satisfied. By completeness, we that the planner is capable of finding all plans which will solve a certain problem.

In order to show that a planner produces correct plans, it is sufficient to show that the planner's truth criterion is sound and that the planner correctly applies that truth criterion. The truth criterion which SPAR uses to determine when operational goals are satisfied is as follows:

An operational goal, g, is satisfied at time t if the following two conditions hold:

1: There is some action, E, in the plan which establishes g and, E is constrained to occur prior to t.

2: For any action, C, which possibly clobbers g, either C occurs after t or there is some action, W, which establishes g and is constrained to occur prior to t and after the execution of C.

In SPAR, the initial state of the world is represented by the add list of a null action. Therefore, in condition 1, E could also be the null action which defines the initial state.

This criterion for satisfying operational goals is sufficient, but it is not necessary. That it is not necessary follows from the fact that it is a restriction of the truth criterion presented by Chapman (and proven to be correct) in his TWEAK planning system. In SPAR, we have made two simplifications from Chapman's truth criterion, both of which limit the method of declobbering goals. First, as we have discussed earlier, separation (i.e. declobbering by constraining the goal not to codesignate with any item in the clobberer's delete list) is not used in SPAR. We have also simplified the addition of a white knight, so that the white knight always reestablishes the goal. In Chapman's truth criterion, the white knight was only required to reestablish the goal when C necessarily clobbered it. As Chapman pointed out, it is very difficult to construct a strict implementation of this condition.

By making these changes, our truth criterion is simpler to implement, but also stricter. Therefore, it is possible for a goal to pass Chapman's test, but fail SPAR's. However, our restrictions do not allow a goal to pass SPAR's test when it would fail Chapman's. This is because we still require establishment and declobbering, we have merely restricted the number of methods the planner can use to declobber the goal. For this reason, it is possible to apply Chapman's correctness proof to SPAR to show that the truth criterion is valid. It is clear (as it is in TWEAK) that SPAR's operational planning implements this truth criterion, since the top level planner is merely a procedural interpretation of that criterion.

For geometric goals, the truth criterion above is modified to include the condition that the geometric constraint be consistent with the constraint network.

A geometric goal, g, consisting of a set of operational goals, O, and a geometric constraint, G, is satisfied at time t if the following two conditions hold:

1: Each operational goal in O is satisfied in the plan.

2: The geometric constraint G is consistent with the plan's constraint network.

Condition 1 simply invokes the truth criterion for operational goals to satisfy the set of goals in O, and condition 2 ensures that the geometric constraint, G, can be added to the plan without resulting in an inconsistent constraint set. Since the planner uses the CMS to determine whether or not geometric constraints are consistent with the current constraint set, in order to determine if SPAR adequately implements this criterion, we must discuss the correctness of the CMS. This was a problem which Chapman did not need to consider in great detail, since his CMS only considered codesignation and ordering constraints.

As we discussed in Section 5, in many cases the CMS does not use exact methods, and approximations to actual solutions are used. On the surface, this would appear to indicate a lack of correctness on the part of the CMS. However, we must recall that correctness only implies that plans which SPAR creates are correct - not that all correct plans can be created. Therefore, as long as the CMS errs on the conservative side, SPAR's correctness will not be jeopardized. That is, correctness is not affected if the CMS declares goals to be false when they are really true. The only way to negatively affect correctness is to declare that false goals have been satisfied. Because of this, all approximations made by SPAR's CMS are on the conservative side. For example, SUP might not find an exact upper bound on a symbolic expression, but it will never find an upper bound which is lower than the exact upper bound. Similarly, the CMS might decide that a grasp cannot be performed when it actually could be performed, but it will never declare that an unreachable grasp can be performed. Thus, we can conclude that SPAR adequately implements the truth criterion for geometric goals.

The truth criterion for uncertainty-reduction goals is quite simple. Since uncertainty-reduction planning examines each action in a specific plan instance, all that is required is to examine the world description just prior to the action to see if it satisfies the uncertainty-reduction goals of the action. However, recall that SPAR does not force the uncertainty-reduction goals to be satisfied. Therefore, it is possible that SPAR will create plans which do not satisfy all uncertainty-reduction goals. We do not consider this to reflect on SPAR's correctness, since SPAR does not claim that such plans are correct, and in fact, takes steps to prepare for errors which might result.

SPAR is not complete, in the sense that SPAR is not guaranteed to find all plans that will solve a particular problem. There are two reasons for this, which have already been mentioned. First, separation to declobber a goal is not implemented in SPAR. Second, SPAR's CMS uses a number of approximations when evaluating constraints. The first of these is not a serious problem. Whenever SPAR fails to find a plan due to its inability to enforce a separation constraint, it can find some correct plan by the introduction of actions to reestablish the goal which was clobbered. Approximating solutions in the CMS is a bit more serious, since it is possible that SPAR could fail to create a plan to solve a problem if the only solution to that problem depended on a plan variable having an instantiation which was not included in the CMS's approximation for that variable's valid values. Of course this is not a serious problem when algebraic constraints are approximated, since they are used only in uncertainty-reduction planning, and when SPAR fails to satisfy an uncertainty-reduction goal it continues with planning while noting that the goal was not satisfied. It can be a problem when planning grasping configurations, if the only valid configuration was not included in the approximation. We assume that such cases will be rare, since they are likely to occur only when the robot is operating very near the boundaries of its work envelope.

7.2. Time Complexity

One of the important results of Chapman's work with TWEAK was his ability to show that the truth criterion could be evaluated in time which was polynomial in the number of steps in the plan. He also proved that planners whose representations are sufficiently complex cannot make this claim. The complexity of SPAR's representations seems to indicate that SPAR's truth criteria cannot hope for a polynomial time implementation. However, as in TWEAK, in order to evaluate the truth criteria for both operational and geometric goals, SPAR also only needs to examine the triples of the form <t,C,W> (where t is the time at which the goal must be satisfied, C is a plan action which might clobber the goal, and W is a possible white knight), since geometric goals are satisfied by constraining the execution of actions which achieve operational goals. (It should be noted that SPAR's representation allows two actions to work synergistically to achieve a goal - for example the assemble action's mate_reachable precondition - but that this synergy is limited to a constant number of actions, so the complexity will remain polynomial even though the exponent will increase.)

In order to claim polynomial time complexity, we must also evaluate the complexity of evaluating constraints, since the CMS is called upon during the evaluation of the truth criteria. As we discussed in Section 5, in some cases our CMS uses a depth first search in order to find an instantiation of plan variables which simultaneously satisfies a set of interacting constraints. Thus, if there interacting constraints on N plan variables, each with m values, the worst case time complexity is $O(N^m)$. Fortunately, because of the nature of the domain, we do not expect N to grow to be very large. The reason for this is that actions in a robot assembly plan tend to have short term effects. For example, variables which are important when grasping an object become irrelevant once that object has been placed in its destination. Since SPAR's plans primarily consist of grasping and placing, it is generally true that N will remain small.

The time complexity for evaluating uncertainty-reduction goals is primarily dependent on the complexity of the algebraic simplifier, the routines SUP and INF, and the number of terms in the resulting constraint. The nature of the simplifier and the routines SUP and INF discourage a rigorous complexity analysis, but we can state that the complexity is dependent on the number of terms in the input expressions. As we have discussed earlier, the worst case number of terms in such an expression grows exponentially with the number of actions in the plan. Therefore, the worst case complexity of evaluating an uncertainty-reduction goal is exponential in the number of actions in the plan.

7.3. Motion Planning

One of SPAR's main limitations at this point is the lack of either gross or fine motion planning. We do not see this as a serious problem, however, because the plans which SPAR constructs are suitable for input to these types of planners. Specifically, SPAR's plans include a set of specifications on the geometric configurations of the end points of actions (i.e. the geometric configuration just prior to and just after the execution of the action). These include specifications of grasping configurations, manipulator configurations and positions for objects. Such specifications could be converted into goal states in a configuration space which could then be used by the motion planning modules. This modular approach to motion planning is similar to the approach used in the TWAIN system [23].

8. CONCLUSIONS

In this paper, we have presented a step toward a planning system which can create assembly plans given as input a high level description of assembly goals, geometric models of the components of the assembly, and a description of the capabilities of the work cell (including the robot and the sensory system). The resulting planner, SPAR, reasons at three levels of abstraction: the operational level (where high level operations are planned), the geometric level (where geometric configurations of the actions are planned) and the uncertainty-reduction level (where world uncertainties are taken into account).

At the first two levels of planning, we have extended the constraint posting approach to domain independent planning by adding geometric preconditions to the actions, linking these to operational goals via plan variables, and expanding the CMS to be able to deal with geometric constraints. At the uncertainty-reduction level of planning, we have expressed uncertainties in the world in terms of homogeneous transformations whose elements are defined in terms of symbolic uncertainty variables. We then expressed limits on tolerable uncertainties in terms of operations on transformations. When the uncertainty-reduction goals cannot be satisfied, rather than abandon the plan, our system augments the plan with sensing operations for verification, and when possible, with local error recovery plans.

At this point, there are a number of areas in our system which are either ad hoc, or require far too much input from the user. For example, the local error recovery plans must be entered by the user, and associated with the uncertainty-reduction goals a priori. One goal of our future work will be to automate this process by employing geometric reasoning about possible errors and error recovery. A further shortcoming of SPAR is the lack of any sort of motion planning system. Incorporating a motion planner with the current system is another goal of our future work.

REFERENCES

[1] A. P. Ambler and R. J. Popplestone, "Inferring the Positions of Bodies from Specified Spatial Relationships," *Artificial Intelligence*, Vol. 6, 1975, pp. 157-174.

[2] W. W. Bledsoe, "The SUP-INF method in Presburger Arithmetic," U. of Texas at Austin Math. Dept. Memo ATP-18, Dec. 1974.

[3] M. Boyer and L. K. Daneshmend, "An Expert System for Robot Error Recovery", Computer Vision and Robotics Laboratory of McGill University, Tech. Report TR-CIM-87-18, Oct. 1987.

[4] R. A. Brooks, "Symbolic Reasoning Among 3D Models and 2D Images," *Artificial Intelligence*, Vol. 17, 1981, pp. 285-348.

[5] R. A. Brooks, "Symbolic Error Analysis and Robot Planning," *The Int'l Journal of Robotics Research*, Vol. 1, No. 4, Winter 1982.

[6] R. C. Brost, "Planning Robot Grasping Motions in the Presence of Uncertainty," Computer Science Department of Carnegie-Mellon University Technical Report CMU-RI-TR-85-12, July 1985.

[7] D. Chapman, "Planning for Conjunctive Goals," *Artificial Intelligence*, Vol. 32, No. 3, July 1987, pp. 333-378.

[8] E. Davis, "Constraint Propagation with Interval Labels," *Artificial Intelligence*, Vol. 32, No. 3, July 1987, pp. 281-331.

[9] R. Dechter and J. Pearl, "Network-Based Heuristics for Constraint-Satisfaction Problems," *Artificial Intelligence*, Vol. 34, No. 1, Dec. 1987, pp. 1-38.

[10] B. R. Donald, "Robot Motion Planning with Uncertainty in the Geometric Models of the Robot and Environment: A Formal Framework for Error Detection and Recovery," *Proc. of the IEEE Int'l Conf. on Robotics and Automation*, 1986, pp. 1588-1593.

[11] B. R. Donald, "A Search Algorithm for Motion Planning with Six Degrees of Freedom," *Artificial Intelligence*, Vol. 31, No. 3, March 1987, pp. 295-353.

[12] H. F. Durrant-Whyte, "Uncertain Geometry in Robotics," *The IEEE Journal of Robotics and Automation*, Vol. 4, No. 1, Feb. 1988, pp. 23-31.

[13] M. A. Erdmann, "On Motion Planning with Uncertainty," MIT AI Lab Tech. Report AI-TR-810, 1984.

[14] R. E. Fikes, P. E. Hart and N. J. Nilsson, "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, Vol. 3, 1972, pp. 251-288.

[15] R. E. Fikes and N. J. Nilsson, "STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol. 2, 1971, pp. 189-208.

[16] M. Gini, R. Doshi, M. Gluch, R. Smith and I. Zualkernan, "The Role of Knowledge in the Architecture of a Robust Robot Control," *Proc. of the IEEE*

Int'l Conf. on Robotics and Automation, 1985, pp. 561-567.

[17] S. N. Gottschlich and A. C. Kak, "A Dynamic Approach to High Precision Parts Mating," *Proc. of the IEEE Int'l Conf. on Robotics and Automation*, pp. 1246-1253, 1988. [An updated version of this paper will appear in a forthcoming issue of *IEEE Trans. on Systems, Man and Cybernetics]*.

[18] S. A. Hutchinson and A. C. Kak, "Applying Uncertain Reasoning to Planning Sensing Strategies in a Robot Work Cell with Multi-Sensor Capabilities," *Proc. of the IEEE Symposium on Intelligent Control*, 1988.

[19] S. A. Hutchinson, R. L. Cromwell and A. C. Kak, "Planning Sensing Strategies in a Robot Work Cell with Multi-Sensor Capabilities," *Proc. of the IEEE Int'l Conf. on Robotics and Automation*, 1988, pp. 1068-1075.

[20] R. E. Korf, "Planning as Search: A Quantitative Approach," *Artificial Intelligence*, Vol. 33, No. 1, Sept. 1987, pp. 65-88.

[21] C. S. G. Lee and M. Ziegler, "A Geometric Approach in Solving the Inverse Kinematics of PUMA Robots," *Proc. of the Thirteenth Int'l Symposium on Ind. Robotics and Robots 7*, Chicago IL, April 1983.

[22] T. Lozano-Perez, "A simple Motion-Planning Algorithm for General Robot Manipulators," *The IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 3, June 1987, pp. 224-239.

[23] T. Lozano-Perez and R. A. Brooks, "An Approach to Automatic Robot Programming," MIT AI Lab, AIM 842, 1985.

[24] T. Lozano-Perez, J. L. Jones, E. Mazer, P. A. O'Donnell, W. E. L. Grimson, P. Tournassound and A. Lanusse, "Handey: A Robot System that Recognizes, Plans and Manipulates," *Proc. of the IEEE Int'l Conf. on Robotics and Automation*, 1987.

[25] T. Lozano-Perez, M. T. Mason and R. H. Taylor, "Automatic Synthesis of Fine-Motion Strategies for Robots," *The Int'l Journal of Robotics Research*, Vol. 3, No. 1. Spring 1984.

[26] G. H. Morris, "Robotic Assembly by Constraints," *Proc. of the IEEE Int'l Conf. on Robotics and Automation*, 1987.

[27] A. Newell and H. A. Simon, "GPS, a Program that Simulates Human Thought," in *Computers and Thought*, eds. E. A. Feigenbaum and J. Feldman, McGraw-Hill Book Co., Inc., NY, 1963, pp. 279-293.

[28] S. Y. Nof, O. Z. Maimon and R. G. Wilhelm, "Experiments for Planning Error-Recovery Programs in Robotic Work," Purdue U. School of Industrial Engineering Research Memo No. 87-2, March 1987.

[29] J. Pertin-Troccaz, "On-Line Automatic Robot Programming: A Case Study in Grasping," *Proc. of the IEEE Int'l Conf. on Robotics and Automation*, 1987.

[30] J. Pertin-Troccaz and P. Puget, "Dealing with Uncertainties in Robot Planning Using Program Proving Techniques," *Proc. of the Fourth Int'l Symposium of*

Robotic Research, Aug. 1987.

[31] R. J. Popplestone, A. P Ambler and I. M. Bellos, "A Language for Describing Assemblies," *The Industrial Robot*, Sept. 1978, pp. 131-137.

[32] R. J. Popplestone, A. P Ambler and I. M. Bellos, "An Interpreter for a Language for Describing Assemblies," *Artificial Intelligence*, Vol. 14, 1980, pp. 79-107.

[33] E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, Vol. 5, 1974, pp. 115-135.

[34] E. D. Sacerdoti, *A Structure for Plans and Behavior*, Elsevier North-Holland, Inc., New York, 1977.

[35] R. E. Shostak, "On the SUP-INF Method for Proving Presburger Formulas", *Journal of the ACM*, Vol. 24, No. 4, Oct. 1977, pp. 529-543.

[36] R. C. Smith, P. Cheeseman, "On the Representation and Estimation of Spatial Uncertainty," *The Int'l Journal of Robotics Research*, Vol. 5, No. 4, Winter 1986.

[37] R. E. Smith and M. Gini, "Reliable Real-time Robot Operation Employing Intelligent Forward Recovery," *Journal of Robotic Systems*, Vol. 3, No. 3, 1986, pp. 281-300.

[38] S. Srinivias, "Error Recovery in Robots Through Failure Reason Analysis," *Proc. of the AFIPS National Computer Conference*, 1978, pp. 275-282.

[39] M. Stefik, "Planning with Constraints (MOLGEN: Part 1)," *Artificial Intelligence*, Vol. 16, 1981, pp. 111-140.

[40] M. Stefik, "Planning and Meta-Planning (MOLGEN: Part 2)," *Artificial Intelligence*, Vol. 16, 1981, pp. 141-170.

[41] P. Tournassound and T. Lozano-Perez, "Regrasping," *Proc. of the IEEE Int'l Conf. on Robotics and Automation*, 1987, pp. 1924-1928.

[42] D. E. Wilkins, "Representation in a Domain-Independent Planner," *Proc. Eighth Int'l Joint Conf. Artificial Intelligence*, 1983, pp. 733-740.