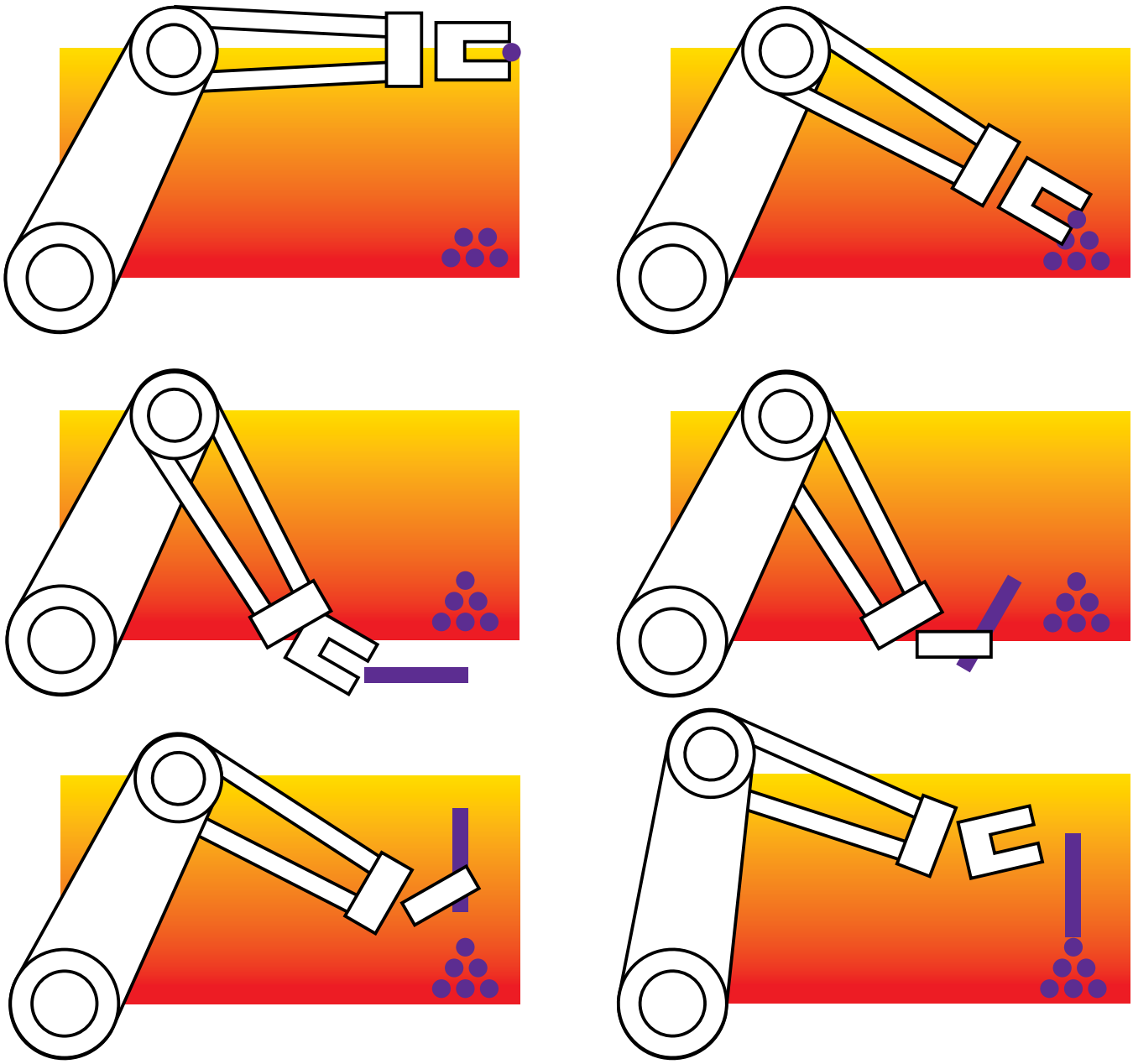
AI Magazine Volume 11 Number 1 (1990) (© AAAI)

Spar: A Planner That Satisfies Operational and Geometric Goals in Uncertain Environments

Seth A. Hutchinson and Avinash C. Kak



In this article, we present Spar (simultaneous planner for assembly robots), an implemented system that reasons about high-level operational goals, geometric goals, and uncertainty-reduction goals to create task plans for an assembly robot. These plans contain manipulations to achieve the assembly goals and sensory operations to cope with uncertainties in the robot's environment. High-level goals (which we refer to as operational goals) are satisfied by adding operations to the plan using a nonlinear, constraint-posting method. Geometric goals are satisfied by placing constraints on the execution of these operations. If the geometric configuration of the world prevents this, Spar adds new operations to the plan along with the necessary set of constraints on the execution of these operations. When the uncertainty in the world description exceeds that specified by the uncertainty-reduction goals, Spar introduces either sensing operations or manipulations to reduce this uncertainty to acceptable levels. If Spar cannot find a way to sufficiently reduce uncertainties, 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.

A prerequisite for intelligent behavior is the ability to reason about actions and their effects. This ability is the essence of the classical AI planning problem in which plans are constructed by reasoning about how available actions can be applied to achieve various goals. For this reasoning process to occur, the planner must be aware of its available actions, the situations in which they are applicable, and the changes affected in the world by their execution. Classical AI planners typically use a high-level, symbolic representation of actions (for example, well-formed formulas from predicate calculus). Although this type of representational scheme is attractive from a computational standpoint, it cannot adequately represent the intricacies of a domain that includes complex actions, such as robotic assembly (consider, for example, that any geometric configuration of the robotic manipulator is a rather complex function of six joint angles). To further complicate matters, there are often uncertainties in the planner's description of the work cell. Therefore, the planner must also be able to reason about the uncertainties in its description of the world and how these uncertainties affect the execution of the actions. For these reasons, classical AI planning methods alone prove inadequate in the domain of robotic assembly.

Spar (simultaneous planner for assembly robots) is a working planner that extends classical AI planning methods to create plans for automated assembly tasks that are to be executed in uncertain environments. In its current implementation, when provided with a description of the initial state of the world and a rudimentary motion-planning algorithm, Spar is able to create plans at a sufficiently detailed level to allow their execution in an actual robotic work cell (a number of the plans that have been created and executed in our lab using a Puma 762 robot arm are presented in Experimental Results). These plans include the geometric specifications of assembly operations (for example, the relative destination positions of objects that are to be assembled), geometric descriptions of grasping configurations (that is, the position and orientation of the robotic manipulator relative to the object that is to be grasped), and geometric descriptions of how objects must be placed in the world (for example, when the robot is used to move an object to some destination position, Spar determines the destination orientation of the object). Spar also plans manipulations to reposition objects if they are initially in positions that are unsuitable for the assembly operations and actions to regrasp an object if the object's initial position does not allow the planned grasping operation to be performed (for example, if one of the object faces to be grasped is in contact with the table, the object must be placed in an intermediate position from which the grasp can be performed). Finally, Spar uses its knowledge about the uncertainty in the world description to assess the possibility of run-time errors. This information is used to add sensing to the plan to reduce uncertainties or, if the resulting uncertainty is still too large, to post verification sensing operations and error-recovery plans. We should note that in the current implementation, Spar's knowledge of the uncertainties in the world description is based on rough estimates of the actual uncertainties because we have not yet performed a rigorous precision analysis of either the sensing system or the manipulator position control.

To create the type of plan described, Spar's approach is to create a plan containing high-level operations (for example, pickup part-1) and then add constraints on the way these operations are executed so that geometric goals are satisfied. It is also possible for operations to be added to the plan to satisfy geo-

0738-4602/89/$4.00 ©1990 AAAI

Spar is a working planner that extends classical AI planning methods to create plans for automated assembly tasks that are to be executed in uncertain environments.

metric goals, for example, if a work piece must be repositioned so that an insertion operation can be performed. To satisfy uncertainty-reduction goals, Spar evaluates the uncertainty in its world description. If this uncertainty is too large to ensure successful execution of an 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 attempt 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 to develop assembly task plans that include geometric descriptions of the actions and sensing operations to reduce uncertainty and verify actions that might not succeed.

These three types of goals (operational, geometric, and uncertainty-reduction) define a three-level planning hierarchy. At the operational level, only a coarse description of the world is used to formulate a high-level plan. As Spar moves down the hierarchy, the detail in the world description increases, first to include the geometric aspects of the world description and finally to include the uncertainty in the world description. Although these levels in Spar bear names that are related to the domain of robotic assembly, we feel that they capture a natural division that could be applied to many planning problems. The top level effectively determines a set of operations that must be performed. The second level constrains how these operations will be performed in the context of domain-specific requirements (in the case of robotic assembly, this performance amounts to constraining the geometric configurations of the robot). Finally, once complete plans have been developed for an ideal world, the uncertainties in the planner's world description are considered.

There are certain limitations to Spar's planning abilities in its current implementation. First, Spar only considers the end points of actions. Thus, if the plan calls for grasping an object and moving it to another place on the work table, Spar determines a set of constraints on the configuration used to grasp the object and the configuration used to place it on the table, but it does not plan the motions required to move the manipulator from the first position to the second. (In our experiments, a rudimentary approach to motion planning was implemented, but this approach is not sufficient for planning motions in complex environments.) Spar also lacks a fine-motion planner. As a result, in some situations where compliant motion could be used to robustly perform an assembly task, Spar pessimistically declares that uncertainties are too great to guarantee successful assembly and that error-detection sensing should be used at execution time. Research is being done in our lab on compliant motion planning (Gottschlich and Kak 1989) that at some future time could be integrated with Spar to address this problem.

A second limitation to Spar is the method used to efficiently deal with the geometric aspects of assembly planning. As is described in Constraints at the Geometric Level of Planning, geometric configurations (for example, grasping configurations) are grouped into equivalence classes, and these classes are assigned labels. This approach allows Spar's constraint-manipulation system (CMS) to treat plan-variable assignment as an instance of the consistent labeling problem. Of course, our goal is to define equivalence groups that preserve the salient distinctions between geometric configurations and ignore distinctions that are not relevant to the assembly planning process. However, this goal is often difficult to achieve, especially because which distinctions are important depends on the task to be performed.

Finally, at this point in time, Spar must know a priori about the locations and orientations of all the objects that participate in the assembly. Therefore, to be used in a real assembly cell, Spar must be augmented with a sensing system capable of determining the positions of the objects to be manipulated. The development of such a system is the objective of a number of research efforts in our lab; see, for example, the work described by Chen and Kak (1989) and Hutchinson and Kak (1989).

The limitations enumerated here, particularly the first two, might cause the reader to

ask why we should even attempt to use classical AI planning techniques for robotic assembly planning. Would it not be more effective, for example, to use a specialized module to derive a high-level task plan from a computer-aided design (CAD) model of the assembly and then use another specialized module to construct motion plans for these actions? Our answer to this question is that to effectively deal with a dynamic world, a planning system must be able to reason about goals and how the actions in a plan work to achieve these goals. For example, if certain goals are already satisfied in the world state, the planner should use this knowledge and not plan for their achievement. Furthermore, plans derived offline (using a CAD model of the assembly) will not always be applicable in an arbitrary world state. Finally, to respond to problems that might arise during plan execution, the planner must possess some knowledge of the purpose that the actions serve, that is, the goals that they are to accomplish. For these reasons, we chose to start with a domain-independent planner and add components to deal with some of the domain-specific issues relevant to robotic assembly planning.

In this article, we present an overview of Spar. Reports of this work can be found in Hutchinson and Kak (1988). The remainder of the article is organized as follows: In Related Research, we review some of the research relevant to the development of Spar. In Planning in Spar, we overview the system, including the top-level search strategy used to satisfy system goals. In Representational Issues in Spar, we describe the representations that Spar uses for uncertainty, plans, actions, and goals. Goal Satisfaction is a description of how Spar satisfies individual goals and includes discussions on the satisfaction of high-level operational goals, geometric goals, and uncertainty-reduction goals. In Constraint Manipulation, we discuss how Spar represents and manipulates constraints. A Task Planning Example brings the first sections of the article together with a detailed example of Spar constructing an assembly plan. Experimental Results includes descriptions of the assembly experiments that have been performed using Spar as the task planner. Finally, Conclusions contains a summary and allusions to future efforts.

Related Research

Research applicable to robotic assembly planning can be broadly divided into two groups: (1) problems that are specifically related to robotic assembly (Donald 1987; Hwang and Ahuja 1988; Liu and Popplestone 1989; Lozano-Perez, Mason, and Taylor 1984) and (2) domain-independent planning (Chapman 1987; Fikes and Nilsson 1971; Sacerdoti 1977; Stefik 1981; Wilkins 1983). The former group includes planners that generate task sequences, planners that automatically derive geometric specifications of assemblies from CAD descriptions, motion planners, and error-recovery planners. The latter group contains the classical AI planners, where the emphasis is placed on the domain-independent aspects of the reasoning process used to develop plans. Neither of these planning approaches has yet produced a planner capable of creating complete assembly plans from high-level specifications of assembly goals. The domain-independent planners lack the ability to reason about geometric concerns and uncertainties in the work cell, and the task-specific planners typically have a narrow focus and deal only with limited aspects of assembly planning. In the paragraphs that follow, we summarize some of the recent research on the various task-specific problems related to robotic assembly planning. Then, we describe some of the limitations of classical AI planners and how Spar overcomes these limitations to effectively deal with assembly planning.

One of the basic problems specific to robotic assembly is the derivation of an ordered sequence of actions that can be used to perform the assembly task. Two approaches to this problem have been investigated. The first approach, discussed in Homem de Mello and Sanderson (1989), uses a graph representation that explicitly contains relationships between subassemblies. A decomposition of the assembly corresponds to a cut set of this graph. Feasible decompositions are used to create an AND/OR graph that represents all valid assembly sequences. A similar approach is described in Huang and Lee (1989), but the end representation of the precedence knowledge is a set of predicates: MP (must precede) and NL (no later than) instead of AND/OR graphs.

A second approach to sequence generation uses a CAD representation for the assembly. With this type of representation, a disassembly sequence is derived, which, when reversed, provides a valid assembly sequence. In this approach, an assembly sequence is generated by first finding mating features of the assembly (see, for example, Liu and Popplestone 1989) and then determining which disassembly operations are valid by using a path-planning approach such as the one described in Hoffman (1989).

. . . motion planning deals with determining a path in free space along which an object can be moved from its initial position to a desired destination.

To generate task sequences, a geometric description of the assembly is required. Rapt (Ambler and Popplestone 1975; Popplestone, Ambler, and Bellos 1978) is a system that automatically derives geometric specifications from symbolic assembly descriptions. The process begins with a set of high-level relationships that must hold in the goal state (for example, faces of two objects are coplanar). Rapt then manipulates the equations that correspond to these relationships to derive a set of homogeneous transformations that represent the goal relationships between the manipulator and the objects to be manipulated.

A blackboard-based sequence planner is described in Liu (1987). The input to this system is a set of assembly instructions, and the output is a sequence of commands that could be executed by a lower-level controller. The blackboard has three levels. The top level is essentially a metaplanning level that focuses the system's attention on specific goals. The middle level deals with solving goals, posting subgoals, and managing constraints. Finally, the lowest level deals with the domain-specific issues of robotic assembly planning (for example, finding candidate pairs of mating features).

For a robot to execute a sequence of actions, it must be provided with motion commands that will affect these actions. In general, motion planning deals with determining a path in free space along which an object can be moved from its initial position to a desired destination. One approach to this problem is based on configuration space (see, for example, Lozano-Perez 1987). With a configuration space approach, the world representation is transformed by expanding the obstacles and shrinking the object to be moved. This approach transforms the motion-planning problem into the problem of moving a single point through the transformed space. Another approach to the problem uses artificial potential fields (Hwang and Ahuja 1988). With this approach, the manipulator moves in an artificial field of forces from its initial position to its destination (which is represented by an attractive pole in the field), always avoiding obstacles (which are represented by repulsive fields). An overview of the various approaches to motion planning can be found in Schwartz and Sharir (1988).

There are basically two approaches to dealing with uncertainties in the planner's description of the world. The planner can attempt to anticipate and avoid errors (Brooks 1982; Pertin-Troccaz and Puget 1987), or the planner can ignore uncertainties and concentrate its efforts on error recovery (Boyer and Daneshmend 1987; Gini et. al. 1985; Gottschlich and Kak 1989; Nof, Maimon, and Wilhelm 1987; Smith and Gini 1986; Srinivas 1978). Consideration of uncertainty in fine-motion planning was discussed in Brost (1985), Donald (1986), and Erdmann (1984). A number of schemes for representing uncertainty in robotic systems have been described (Donald 1987; Durrant-Whyte 1988; Smith and Cheeseman 1986), but these are not currently part of a planning system.

We are aware of two systems that have a somewhat larger scope than those listed previously: Twain (Lozano-Perez and Brooks 1985) and Handey (Lozano-Perez et. al. 1987). These planners begin with a high-level task plan and then add motion plans for the individual actions in this plan. Twain is a constraint-posting planner that can also add sensory operations to the plan to reduce uncertainties. Handey is an integrated system that includes a sensory system to determine the initial world state. One of Handey's main strengths is its ability to plan grasping operations when the objects are in cluttered environments (Tournassound and Lozano-Perez 1987). It should be noted, however, that neither Twain nor Handey is capable of reasoning about the effects of actions and how they might be used to achieve goals.

The planner we describe in this article combines domain-independent planning techniques with a number of modules containing task-specific knowledge. This arrangement allows Spar to use a nonlinear constraint-posting approach as its top-level control structure and domain-specific knowledge to evaluate constraints during planning. Using a constraint-posting approach, Spar seeks to satisfy a goal by first examining all the actions and constraints previously generated to see if the goal can be satisfied by merely adding a new constraint (where a *constraint* can be viewed as a specification or a restriction on an action). If this strategy fails, a new action is added to the plan. Nonlinear planning allows actions to be added to the plan without imposing a strict ordering on the set of actions.

One of the primary drawbacks of traditional domain-independent planners, for example, Sipe (Wilkins 1983) and Tweak (Chapman 1987), is that the representations used must be domain independent. Thus, these planners use high-level symbolic constructs to represent the world and the effects of actions. For example, both Sipe and Tweak use well-formed formulas (WFFs) from predicate calculus to represent the effects of actions. In a complex domain, such as robotic assembly,

We maintain that Spar's three-level hierarchy (operational, geometric, and uncertainty-reduction) is a natural division for many domains.

this type of high-level representation is not sufficient to represent all relevant aspects of the world. For example, although it is possible to represent a number of spatial relationships with WFFs (for example, on(block1, block2)), there would be no way to describe the reachable configurations of the robot arm with such a high-level representation because the reachable configurations are defined by a subset of an N-dimensional continuous space (for a robot with N joints). Therefore, although Sipe easily solves high-level blocks world problems (such as "put some blue block on the top of some green block"), it is not capable of solving the lower-level details of these problems (for example, determining the joint angles that must be used to position the robot arm to perform the stacking operation).

To cope with these problems, Spar extends the planning beyond higher-level symbolic goals (which we refer to as operational goals) to include geometric and uncertainty-reduction goals. To plan with these additional goals, we added CMS that contains domain-specific knowledge (including the kinematics of the robot, object models, and sensing operations). This domain knowledge is used by CMS to determine whether constraints on such elements as robot arm configurations can be satisfied.

It should be noted that a number of domain-independent planners allow for a hierarchical representation of the world (for example, Sipe); however, it is our contention that the lowest levels of representation in these planners are still not capable of representing all the relevant details of complex domains, such as robotic assembly (as we described earlier). We maintain that Spar's three-level hierarchy (operational, geometric, and uncertainty-reduction) is a natural division for many domains. The highest level is useful for creating plans that do not account for the intricacies of the low-level domain constraints. At the second level, the domain constraints not amenable to symbolic representation are taken into account (geometric constraints in our domain) by using domain-specific components of CMS to assess constraint satisfiability. Finally, a third level is used to take into consideration the uncertainties in the world representation. At this point, we should note that Spar is not conceptually limited to three levels. Any of the levels could be expanded to include multiple levels of abstraction if this expansion were appropriate for the domain (for example, the top level could be broken into a number of levels corresponding to a hierarchical organization of operations).

Planning in Spar

To create complete assembly plans, we extended the planning that is done in traditional constraint-posting planners, such as those described in Chapman (1987) and Wilkins (1983), to include both geometric planning and uncertainty-reduction planning. By *geometric planning,* we mean the planning that determines the actual geometric configurations that 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 that are used to manipulate objects. *Uncertainty-reduction planning* involves first determining whether the uncertainty in the planner's description of the world (for example, the possible error in part locations) is sufficiently small to allow plan execution to succeed. If the uncertainties are too large, then sensing operations or manipulations are added to the plan in an attempt to reduce the uncertainty to an acceptable level. If this approach 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 recovery from possible run-time errors. We call Spar a simultaneous planner because all three levels of planning influence one another.

When designing a nonlinear constraint-posting planner, the degree to which constraint posting is used is an issue that must be considered. A pure constraint-posting planner would make no variable instantiations until all its goals had been satisfied, at which time CMS would determine the variable instantiations that simultaneously satisfied all the constraints. The advantage to this approach is a

decrease in the amount of backtracking, which results from avoiding arbitrary choices that 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, because of 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 elaborate on these complexities in Satisfying Uncertainty-Reduction Goals). 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 constraints are 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 goal types; however, for the reasons of complexity that we just mentioned, it is not expedient to 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 because the cost of maintaining the constraint network associated with these two types of goals is significantly less than the cost of a backtracking search algorithm.

The computational savings achieved by dividing Spar's planning into two phases comes at the expense of completeness. It is possible that during the second phase of planning, situations will arise in which the uncertainty in the world description can not be sufficiently reduced without the addition of higher-level plan segments. Consider, for example, a situation in which the robot is to grasp a single object that is located near the center of a number of objects whose positions are known. Furthermore, suppose there is some uncertainty in the position of the target object, such that this uncertainty will not permit the object to be grasped without the possibility of a collision between the manipulator and one or more of the surrounding objects. In such cases, it would be desirable for Spar to invoke the top-level planner to devise a plan to systematically remove the cluttering objects. With our current division of planning into two phases, this approach is not possible. In such cases, Spar would declare that uncertainties in the world description could not be sufficiently reduced to guarantee successful execution of the plan, and verification sensing and a local recovery plan would be added.

It should be noted that Spar is capable of adding actions to the plan in the second phase of planning but only single, special-purpose actions. Although these actions have preconditions, no planning is done to achieve these preconditions. If they are not satisfied by the plan handed down from the first phase of planning, then the uncertainty-reduction action cannot be used.

Spar's planning process begins with a null plan and a set of goals that the user supplies. This null plan is then refined until all goals are satisfied. As described earlier, this process occurs in two phases. First, a constraint-posting approach is used to satisfy all operational and geometric goals. Then, a second phase of planning is used to satisfy the uncertainty-reduction goals.

In the first phase of planning, Spar iteratively refines the current partial plan so that it satisfies some pending goal. This refinement of the current partial plan is done by either constraining the execution of an action that is already in the plan or introducing a new action into the plan. In the latter case, Spar adds the new action's geometric and operational preconditions to appropriate goal stacks and checks each currently satisfied goal, noting those that are possibly undone by the new action and placing them on the appropriate goal stack. The first phase of planning terminates when there are no more pending operational or geometric goals.

In the second phase of planning, Spar does not use the constraint-posting approach. Instead, the uncertainty-reduction preconditions are considered for specific plan instances. 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. Specific plan instances are examined until one is found in which all uncertainty-reduction goals can be satisfied. If no such instance can be found, the instance that contained the fewest unsatisfied uncertainty-reduction goals is selected. This plan instance is augmented with sensing verification actions and potential recovery plans for anticipated possible errors.

Figure 1 shows a block diagram of Spar. To the left are the goal stacks and a set of satisfied goals that are used to track planning

progress. The dashed box at the top represents Spar's knowledge about the possible actions. This knowledge includes the templates that are used to represent actions, a set of rules for instantiating these templates, a set of actions to be used to reduce uncertainty in the world, and a set of procedures that 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 system includes the actual CMS and a number of domain-dependent modules that are used to evaluate constraints. These modules include routines to find upper and lower bounds on symbolic expressions (SUP/INF), an algebraic simplifier (SMP), and routines to solve the inverse kinematics of the robot (IKS). The constraint system also includes a constraint network that is used to organize the plan's constraints. Finally, the bottom of the figure depicts the output of the planner: the verification sensory operations and local recovery plans (used when uncertainty-reduction goals cannot be satisfied) and the set of actions in the plan.

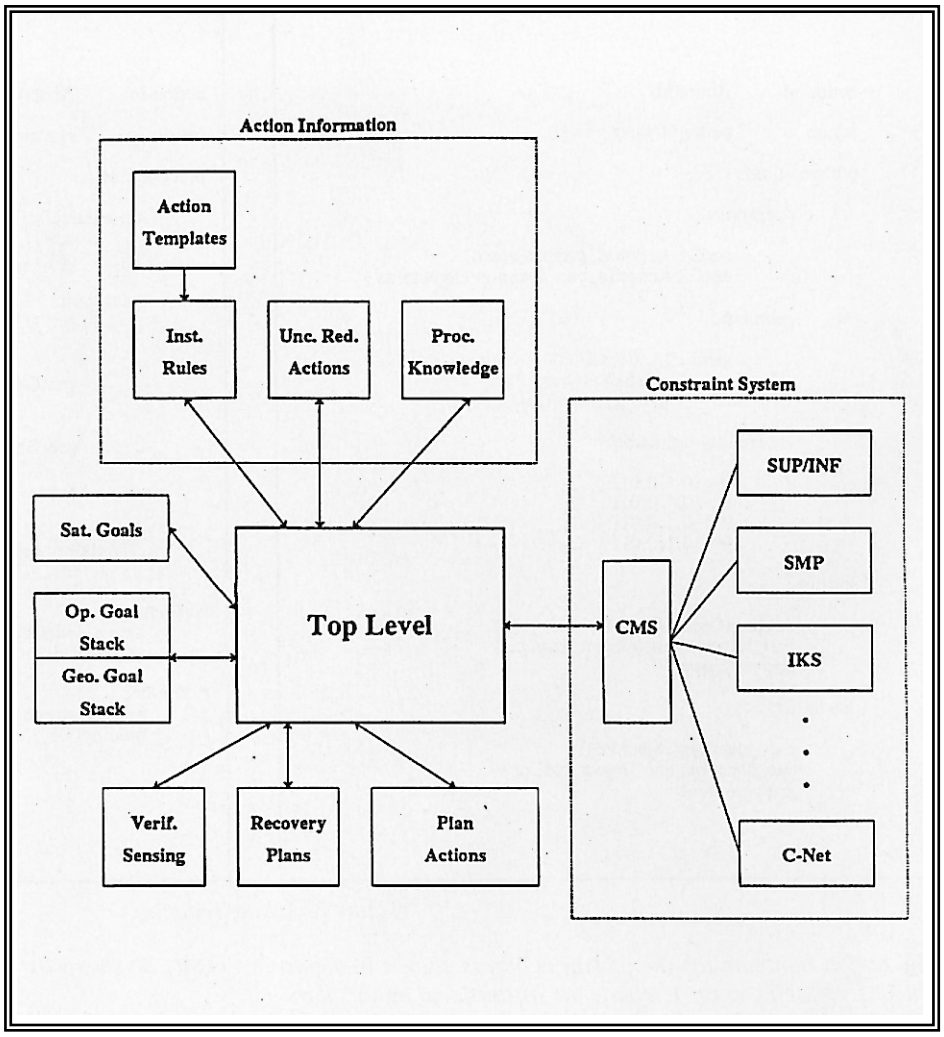


Figure 1. Block Diagram of Spar.

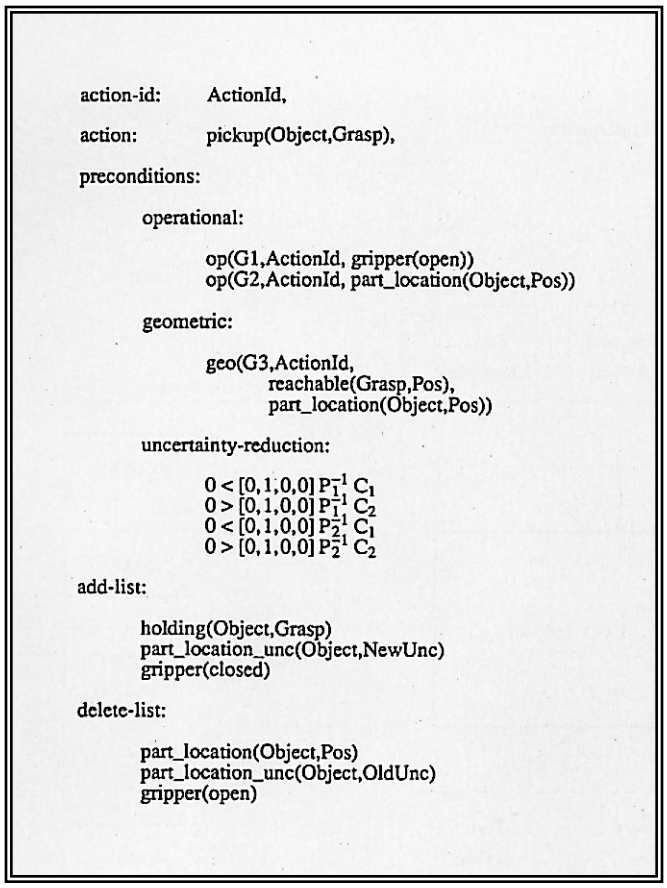
Representational Issues in Spar

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

Representation of Actions

Currently, Spar plans with three actions: pickup, putdown, and assemble.[1] These actions are represented by action templates, each of which has the following five components: (1) action id, an identifier that Spar uses to reference a particular instance of the action; (2) action, the name of the action and its arguments; (3) preconditions, the operational, geometric, and uncertainty-reduction preconditions that must be met prior to executing the action; (4) add list, a list of conditions that will be true in the world after the execution of the action; and (5) delete list, a list of conditions that will no longer be true in the world after the execution of the action.

Figure 2 shows the action templates for the three actions. The meanings of the various preconditions are made clear in subsequent sections of the article. We should also note that the actions currently implemented allow the manipulation of a single object or the assembly of two objects. To facilitate the simultaneous assembly of more than two objects (for example, inserting a cylindrical object into a hole and simultaneously through a washer), additional action templates would have to be added. This restriction does not prohibit Spar from planning



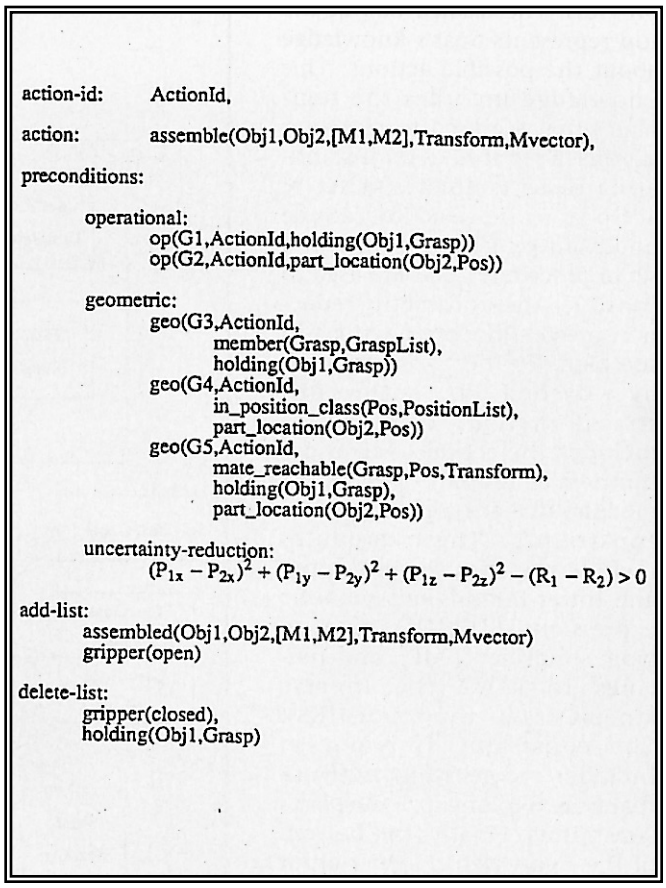


Figure 2. Action Templates.

The action template for the pickup action is shown in figure 2a. The variables in the template are instantiated by the planner using the rule shown in figure 3.

Figure 2b shows the action template for the assemble action.

assemblies with more than two components, only from planning assembly operations in which more than two components are required to participate in a single assembly step.

When Spar adds an action to the plan, it instantiates the template for this action so that it will accomplish the particular goal that caused the action's addition. This process consists of instantiating the various identifiers in the action to unique labels (for example, the ActionId and the Gi's in figure 2) and then either instantiating or constraining the plan variables in the action so that it achieves the goal. Spar uses a set of rules to determine the proper variable instantiations for an action template, given the goal that the action is to achieve. Figure 3 shows an example of the rule that instantiates a pickup action template to achieve the goal holding(Object, Grasp). (Note that the uncertainty-reduction preconditions do not appear in the instantiated template because in Spar's current implementation, 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 operations to specific actions based on the objects that will be manipulated by these actions.

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 figure 3 specifies the initial constraint that Grasp be one of the possible grasps for Object. (We discuss how grasps, stable poses, and so on, are represented in Constraint Manipulation.) Specification of a set of constraints to be satisfied by a plan variable amounts to assigning an initial label set to a node in the constraint network (also discussed in Constraint Manipulation). When initial constraints are added, there is no need to invoke CMS to see if they are consistent with the current constraint network because the variables that will be constrained by these initial constraints did not previously exist in the plan and, thus, did not occur in the constraint network.

```
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 2 (continued). Action Templates.

```
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 3. Rule to Instantiate an Action Template.

Figure 2c shows the action template for the putdown action.

The Representation of Uncertainty in Spar

To create assembly plans that 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 this 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 subsection, we address each of these three issues.

Representing Uncertain Quantities. In our current implementation of Spar, we chose to limit the number of quantities that are considered uncertain. For an object resting on the work table, the *X,Y,Z* location of the object (that is, 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 choice reflects our assumption that objects resting on the work table will be in stable poses, which fix two rotational degrees of freedom of the object (this assumption is discussed further in Constraint Manipulation). 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.

All uncertainties in Spar are expressed in terms of uncertainty variables. The possible values for an uncertainty variable are defined using bounded sets. We represent the uncertainty in the position of an object by a homogeneous transformation matrix whose entries are expressed in terms of uncertainty variables.[2] By combining the *ideal position* of an object (that is, the position of the object if all uncertainty is eliminated) with the uncertainty in this position, we obtain the possible position of an object. This possible position will be a homogeneous transformation matrix, with some or all of its entries expressed in terms of uncertainty variables. Any matrix that can be obtained by substituting valid

. . . SPAR does its planning in two phases. . . a constraint-posting approach is used to satisfy operational and geometric goals; . . . specific plan instances are examined to find plans that satisfy uncertainty-reduction goals.

values for the uncertainty variables will represent one possible position of the object.

Given these assumptions, we define the transformation that 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 the transformations that could be obtained by substituting valid numeric 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 (that is, 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 because of the rotational component in the uncertainty. In particular, the axis of this rotation is not defined by the local frame of the object or the world frame but by the world Z axis, translated to the origin of the object's local frame. We, therefore, separately consider the uncertainty in the displacement of the object and the rotational uncertainty. For the displacement, let the transformation $Tr_{\Delta O}$ be a transformation that defines the uncertainty in the X,Y,Z location of the object relative to the world coordinate frame:

$$Tr_{\Delta 0} = \begin{bmatrix} 1 & 0 & 0 & \Delta X_0 \\ 0 & 1 & 0 & \Delta Y_0 \\ 0 & 0 & 1 & \Delta Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

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, because the rotational uncertainty is about the world Z axis translated to the origin of the object's local frame, it can be represented by postmultiplying the possible object displacement by a rotation about the Z axis, R_{DO}, 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

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

Derivation of Uncertainty-Reduction Goals. To illustrate the construction of the uncertainty-reduction preconditions, in this section, we derive the uncertainty-reduction preconditions for the pickup action. We derive these preconditions 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 (that is, the points on the object that 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

derive these preconditions, 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 $\pm 1/2 W_m$ along the Y axis of this frame (where W_m is the distance between the two fingers) (figure 4). Using equation 1, we find

$$P_1 = T_{M+\Delta}\, trans(0,-1/2W_m,0)$$

$$P_2 = T_{M+\Delta}\, trans(0,+1/2W_m,0),$$

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

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

In this case, the possible position of the manipulator is obtained by using equation 1 but replacing T_M by the composition of the object position (T_O) and the grasping configuration T_G (which expresses the position of the manipulator's coordinate frame relative to the object's local frame).

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 possible 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 (that is, the axis that defines the direction of finger opening and closing). Using equation 2, we find

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

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

where W_g is the width of the object at the grasp point. Note that we are not interested in the coordinate axes at the contact points, only the displacement.

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 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

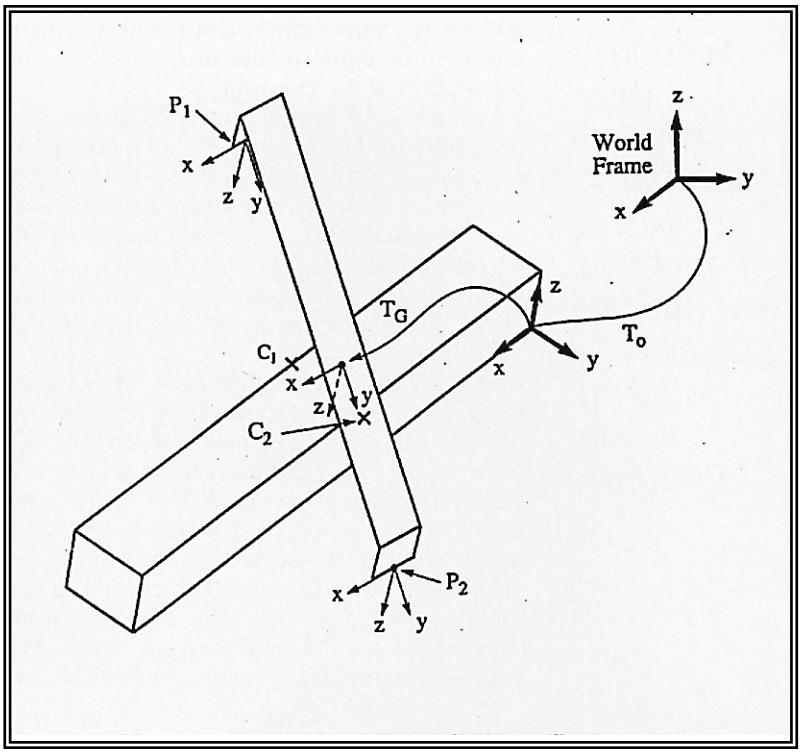


Figure 4. Possible Finger Coordinate Frames and Contact Points for the Pickup Action.

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

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 the matrix multiplications shown must be performed symbolically. That is, the numeric computations will not actually be performed because some of the entries in the expressions will not have specific numeric values (for example, the value of the variable ΔX might only be known to lie somewhere in the interval $[-0.5,0.5]$).

The Propagation of Uncertainty by Actions. To illustrate how actions propagate uncertainty, in this subsection, we describe how the pickup action affects the uncertainties in the position of the object to be grasped. In general, the pickup action has the effect of reducing the uncertainty in the position of the object to be grasped. Uncertainty is reduced because the new uncertainty in the object's position will be defined in terms of the manipulator uncertainty, which is normally less than the uncertainty in positions that are determined by the sensing system. Specifically, the pickup action has the effect of transforming the 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

Goals in Spar have three relevant attributes: a type . . . a condition that must be satisfied . . . and an action identifier.

of the manipulator. The pickup action also reduces the uncertainty in the object's orientation to be equal to the uncertainty in the orientation of the manipulator.

To derive an expression for the reduced uncertainty, let $Tr_{\Delta O}$ be the displacement uncertainty in the object's position just prior to the execution of the pickup action. Therefore, as described in Representing Uncertain Quantities, this uncertainty is defined relative to the world coordinate frame. We need to obtain a displacement error $Tr'_{\Delta O}$ such that

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

where R_M is the transformation that represents only the orientation of the manipulator (that is, 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 the object and R_G to be the transformation that represents the orientation of the manipulator relative to the local frame of the object (that is, the rotational part of T_G), we find

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

given 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 the 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.

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; during the second phase, specific plan instances are examined to find plans that 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 these 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 that can 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 (that is, a set of values for the plan variables that satisfies the constraints in the constraint network) and performing this 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. The error count 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 greatest chance of success.

Representation of Goals

In this subsection, we describe Spar's representation of goals. Because a previous section dealt with Spar's representation of uncertainty and uncertainty-reduction goals, we do not discuss the uncertainty-reduction preconditions of actions here.

Goals in Spar have three relevant attributes: a type (either operational, geometric, or uncertainty-reduction), a condition that must be satisfied (that is, 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 use the terms *goal* and *precondition* to refer to either the condition part of the goal or the entire structure. Which of these is meant should be evident from the context.

Spar's operational goals are similar to the high-level goals used in traditional domain-independent planners (for example, Strips or Tweak). One difference is our inclusion of plan variables that 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 is

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

where holding(Obj,Grasp) is the single operational goal, and reachable(Grasp,Pos) is the geometric constraint.

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.

Table 1. The Constraints That Are Used in the First Phase of Planning.

Goal Satisfaction

In this section, we discuss the methods that Spar uses to satisfy operational, geometric, and uncertainty-reduction goals. We frequently allude to the role of CMS in the process of goal satisfaction, but we leave a detailed discussion of CMS for Constraint Manipulation. For the purposes of this section, it is sufficient to assume that CMS is capable of determining if a new constraint is consistent with the current constraint set.

Satisfying Operational Goals

In Spar, ensuring the satisfaction of an operational goal proceeds in two steps: finding an action that establishes the goal and dealing with actions that could violate (or undo) the goal.

To find an action that establishes an operational goal, Spar first looks at the add lists of the actions that 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 this unification is performed, and the action is declared to have established the goal. If Spar succeeds in finding such an action, this action is constrained to take place prior to the time at which the goal must be satisfied. If CMS determines that this new ordering constraint is not consistent with the current constraint network, the constraint addition fails, and Spar backtracks in an attempt to find another action in the plan that establishes the goal.

If Spar fails to find an action in the plan that can establish the goal, it adds a new action. Adding a new action 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. Any time Spar adds an action to the plan, it is possible that the new action might violate goals that 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 those that could be violated by the new action to the appropriate pending goal stack.

Once an operational goal is established, Spar examines each action in the current partial plan to see if it could possibly violate the goal. An action can violate 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 goal violation: (1) the violating action can be constrained to occur after the time at which the goal must

In Spar, geometric goals are satisfied by constraining the way in which plan actions are performed.

be satisfied (promotion of the goal), (2) the goal can be constrained not to unify with any clause in the violating action's delete list (separation), and (3) an action can be used to reestablish the goal. A reestablishing action can either be an action that is already in the plan or a new action that is specifically added for the purpose of reestablishing the violated goal.

In Spar, it is difficult to add separation constraints to the plan because the unification is done using Prolog's internal unification algorithm, which will not take into account constraints in Spar's constraint network. Therefore, it is difficult to implement a constraint that says 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 omitted separation as a possible means of protecting goals in Spar.

Promotion of the goal is the first option that Spar tries when protecting goals from violation. When an action, *C*, can violate 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 violating action should not be executed until after action *S* has been executed.

Using an action to reestablish a goal is identical to establishing the goal, with the additional condition that the action must occur after the potential violating action. As such, this process proceeds exactly as the establishment process described previously, but when a candidate action is found, the additional constraint prior_to(*C,A*) is added to the constraint network (where *C* is the action identifier of the violating action, and *A* is the action identifier of the new action). Earlier, we mentioned the possibility of constraining the violating action to occur before the establishing action. This process is the same as allowing the establishing action to also act as the reestablishing action.

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 this goal is to place a constraint on how the manipulator performs the grasping action. Thus, Spar needs to link the operational and geometric levels of planning. For this purpose, when planning to satisfy operational goals, plan variables are introduced that can be constrained by the geometric level of planning to determine how an action is executed. The geometric preconditions are expressed in terms of these 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 that 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 was discussed in Representation of Goals, geometric goals consist of a set of operational goals and a geometric constraint that is to be applied to the actions that achieve the operational goals. Each operational goal that is associated with a geometric precondition of an action is also separately listed as an operational precondition of the action. Therefore, because 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, to satisfy a geometric goal, Spar first finds the actions that establish its associated operational goals and attempts to constrain the execution of these actions so that the geometric constraint is satisfied. This is done by instructing CMS to add the geometric constraint to the constraint network. If the added constraint is consistent with the other constraints in the network, the goal is satisfied and moved to the list of satisfied goals.

If 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 Representation of Actions. 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 because the new action will contain new plan variables that 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 violate geometric constraints because 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 violating the new geometric constraint, this violation would have been detected by CMS when attempting the constraint addition.

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, uncertainty-reduction planning does not use the constraint-posting method. Second, if no plan instance can be found that 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 because of the complexities involved with their representation and evaluation. The high cost of representing uncertainty-reduction goals compared to operational or geometric goals is in part because the geometric and operational effects of actions typically do not propagate through more than one action, but uncertainties can propagate through many actions. For example, consider the following sequence of actions:

action1: pickup(part1,grasp1)
action2: putdown(part1,position1)
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, although 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 that 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 The Representation of Uncertainty in Spar, Spar uses symbolic algebraic expressions to represent uncertainty. Each time a plan action affects the uncertainty in some quantity, in the worst case, there is 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. Because CMS uses upper and lower bounding routines to evaluate uncertainty-reduction constraints, and 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 (for example, in evaluating constraints on the robot's joint angles).

There are two reasons for not backtracking into the first phase of planning. First, because Spar represents uncertainty in the world using bounded sets (for example, 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 that 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, although, as discussed in Planning in Spar, there are certain cases where such backtracking would be beneficial.

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 cre-

Spar currently uses three types of constraint . . . ordering constraints . . . binary constraints . . . symbolic algebraic inequalities . . .

ation of an augmented plan instance that contains four components: (1) 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), (2) an error count (initially set to zero), (3) a list of sensory verification actions (initially set to the empty list), and (4) 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 updating 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 this goal. Such algebraic inequalities are derived by performing an appropriate combination of symbolic matrix multiplications, matrix inversions, and so on, as determined by the actual goal. It should be noted that many of the quantities that enter into these operations will be defined in the world state (for example, the part locations, uncertainties in the part locations).

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. Sensing actions have the same representation as manipulations. The add and delete lists of a sensing action template contain elements that 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 expressions for the failed uncertainty-reduction goal.

If the sensing operations fail to reduce the uncertainty to acceptable levels, Spar attempts to introduce manipulations into the plan that can reduce the uncertainty. Currently, the only manipulation that 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 described in The Propagation of Uncertainty by Actions. Because the operational and geometric preconditions for this action are the same as for the pickup action, it can always be spliced into the plan instance just prior to the execution of some existing pickup action; however, the uncertainty in the world description must satisfy the uncertainty-reduction preconditions for the uncertainty-reduction pickup action.

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. Second, a sensing verification action and a local recovery plan are added to their respective lists in the augmented plan instance. 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 that could occur and automatically prescribe verification strategies and recovery plans.

Constraint Manipulation

In Spar, the bulk of the domain knowledge resides in CMS, which allows the top-level planning to proceed without any need to understand the domain of automated assembly. The action descriptions include preconditions on geometric configurations and the tolerable uncertainties in the world description, but to satisfy these preconditions, the top-level planner merely requests that CMS add constraints to the constraint database. It is the task of CMS to determine whether these new constraints are consistent with the current constraints in the plan, which, in turn, requires a certain amount of domain-specific knowledge.

Spar currently uses three types of constraint. In operational planning, *ordering con-*

straints are used to ensure that actions are performed in the proper sequence (and that goals are satisfied at the appropriate times). In geometric planning, *binary constraints* between object positions and manipulator configurations are used 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 that can exist in the world description prior to the execution of an action.

Throughout the previous sections of the article, we referred to 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 because 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 this 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 describe the constraints that are used in Spar, their semantics, and how CMS determines whether 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 constraint set will be determined to be inconsistent when it really isn't. The reason is that the quantities that enter into the constraints in Spar are 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 devised a representation of grasping configurations that approximates the true situation. This arrangement simplifies the process of constraint manipulation but adds the possibility that Spar might overlook certain solutions.

Ordering Constraints in Operational Planning

In the first phase of planning (used to satisfy operational and geometric goals), Spar operates as a nonlinear planner, so 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. Although 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.

Spar's CMS uses a directed graph (which we refer to as the ordering graph) to track 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 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 because the only type of inconsistency that might arise is if an action is constrained to occur both prior to and after some other action in the plan.

Constraints at the Geometric Level of Planning

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

We begin this subsection with an introduction to binary constraint networks and an explanation of how such a network is used to represent Spar's geometric constraints. Then, we describe each type of geometric constraint included in Spar and the mechanisms used to evaluate these constraints.

Stable poses provide a method of specifying destination positions in terms of the object's orientation . . .

The Geometric Binary Constraint Network. This subsection includes a cursory introduction to constraint networks. A more thorough introduction can be found in Davis (1987) or Dechter and Pearl (1987). We begin our discussion with the following definitions:

Definition: The *label set* for a plan variable is the set of possible values that can be assigned to this variable.

Definition: A *unary constraint* on a variable is a restriction of the variable's label set.

Definition: A *binary constraint* on two variables, V_i and V_j, is a relation, $C_{ij} \tilde{A} L_i \times L_j$ where L_i is the label set of V_i, and L_j is the label set of V_j.

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

Spar's CMS uses depth-first search with backtracking to determine network consistency. For each level in the search, one node in the network is selected that has not yet been assigned a value; a value is assigned that 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, so any value from the node's label set can be chosen). The algorithm is similar to that described in Dechter and Pearl (1987).

To represent Spar's geometric constraints using a binary constraint network, each geometric plan variable (for example, 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 that can be assigned to this variable (determined by the action template instantiation rules discussed in Representation of Actions). 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 the grasping configurations for this object (grasping configurations are described in Reachability of Grasps).

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 that 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.

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

Set Membership. 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, CMS adds one and assigns its initial label set to contain the elements of Labels. If a node is already in the constraint network for Variable, 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 (that is, 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 that assign Variable a value that is not in its new label set. The new network is then checked for consistency. If both node and network consistency are satisfied, CMS returns success.

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. Thus, we characterize object positions using equivalence classes. These classes are based on the object's *stable poses,* that is, orientations of the object that allow 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 deter-

mining 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, because stable poses are only used to determine a list of occluded features and specify destinations of held objects, an object not being in a stable pose will not be a problem as long as the sensory system is capable of determining by inspection the object's occluded features.

With 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 that are allowable for a certain assembly operation and constrain the object's position to correspond to one of these 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 that 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, Spar must determine (in one of two ways) which stable pose of the object corresponds to Position. 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 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 (for example, 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 that are occluded. Position is then determined to correspond to the stable pose that obscures the same set of features.

Aside from the situation described in the previous paragraph, 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 that are connected to Position's node, and then checks for network consistency.

Reachability of Grasps. Whenever the planner inserts a manipulation action into the plan, it must ensure that all the configurations required to perform this manipulation will be physically realizable. To guarantee this physical realizability of manipulation, Spar uses two constraints:

reachable(Grasp,Position)

and

mate_reachable(Grasp,Position,T_a) .

The first of these constraints 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 this combination must be physically realizable. This constraint is used in both grasping and placing objects. The second constraint is used for mating operations, where T_a is a homogeneous transformation that 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 for the reachable constraint to be satisfied. First, the faces of the grasped object that 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. Second, the robot must be able to perform the grasp without exceeding any of its physical joint limits and contacting the work table.

For the mate_reachable condition, only the second condition is used. However, the position that 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 the first condition, 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 constructive solid geometry 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 that 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 discussed in Stable Poses and Position Classes.

For specific values of Grasp and Position,

In our system, grasping configurations specify not only the geometric configuration that is used to grasp the object but also the set of object features that are obscured by the grasp . . .

the second condition is verified by invoking routines that compute the inverse kinematic solution for the robot's joint angles given an absolute position of the end effector, as follows: A particular grasp has associated with it a homogeneous transformation that defines the coordinate frame of the robot manipulator relative to the frame of the object. We refer to this transformation as the *grasp transformation*, or T_g. If Position is an absolute position (that is, 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 that are found by this program are then tested to ensure that they are within the robot's limits. 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 Lee and Ziegler (1983). At this time, to prevent collision with the work table, Spar simply forces the *Z* axis of the manipulator to roughly point in the direction of the negative world *Z* axis, which prevents the robot from attempting to reach up through the table to grasp an object.

If Position corresponds to a stable pose (that is, it specifies an orientation of the object but no absolute *X,Y,Z* position), 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 (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 CMS is instructed to add either a reachable or mate_reachable constraint to the constraint network, the two conditions previously described 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). The valid pairs are found by exhaustively pairing every value from the label set for Grasp with every value from the label set for Position and recording all pairs that 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, CMS signals failure, and the old network is restored. Otherwise, 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 described earlier, a finite number of possible stable poses are 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 that are obscured by the grasp and the features of the object that come into contact with the manipulator fingers in the grasp. This approach is similar to that described in Pertin-Troccaz (1987) and Tournassound and Lozano-Perez (1987).

By making this type of quantization of the space of grasping configurations, we replace exact descriptions of grasping configurations with approximations. Thus, it is possible that Spar will occasionally errantly determine that a reachability constraint is not consistent with the current constraint set. In general, we do not expect such a result, except when the manipulations that are to be performed require the robot to operate near the boundaries of its work envelope.

Constraints at the Uncertainty-Reduction Level of Planning

As we described previously, when the planner considers the uncertainty-reduction goals, it does so for a particular plan instance. As a consequence, 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 particular, because the uncertainty-reduction goals are expressed as inequalities of the form $expr_1 < expr_2$ and because at least one of these expressions is always a single constant, if we find the maximum value for $expr_1$ and the minimum value for $expr_2$ (under the constraints contained in the world description), we can determine whether the uncertainty-reduction goals are met simply by checking to see if $max(expr_1) < min(expr_2)$.

To find upper and lower bounds on symbolic expressions, we have implemented a system similar to the SUP/INF system that was introduced by Bledsoe (1975), then refined by Shostak (1977) and later by Brooks

(1981) for his Acronym system. The functions SUP and INF each take two arguments, a symbolic expression and a set of variables, and return upper and lower bounds on the expression in terms of the variables in the variable set. The method SUP/INF uses 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 approach 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. Thus, it is possible that SUP/INF will sometimes find bounds that are not exact.

In spite of this disadvantage, the policy of recursively finding bounds on subexpressions and then combining these bounds guarantees that the algorithms will terminate, as 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 Brooks) that the calculated bounds are conservative, in that SUP always returns a value that 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 sometimes only approximates solutions is not a severe problem for Spar because failure to satisfy uncertainty constraints has the worst-case result of adding sensing actions to the plan. That is, if CMS determines that the uncertainty constraints cannot be satisfied, it does not backtrack. It merely prepares for possible failure.

A Task Planning Example

In this section, we illustrate Spar's flow of control with an assembly example. Consider the task of mating the two objects shown in figure 5. 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 with a goal of the form

```
assembled( peg,
           block,
           [[7,8],[7,8]],
           tm([[-1,0,0], [0,1,0], [0,0,-1],
               [3,0,3.25]]),
           [-4,0,0]) ,
```

where peg and block are the two objects to be assembled (the peg is the held object, and the block is stationary), the third argument contains a list of the mating features for the peg and the block (here, both the peg and the block have surfaces 7 and 8 as mating features), the fourth argument is a homogeneous transformation that expresses the destination position of the peg relative to the local coordinate system of the block (for example, the *X* axis of the peg's local coordinate system will be parallel to the vector [*-1,0,0*] in the block's local coordinate system), and the final argument is the mating vector (that is, the vector along which the mating motion will occur) expressed in the block's local coordinate frame.

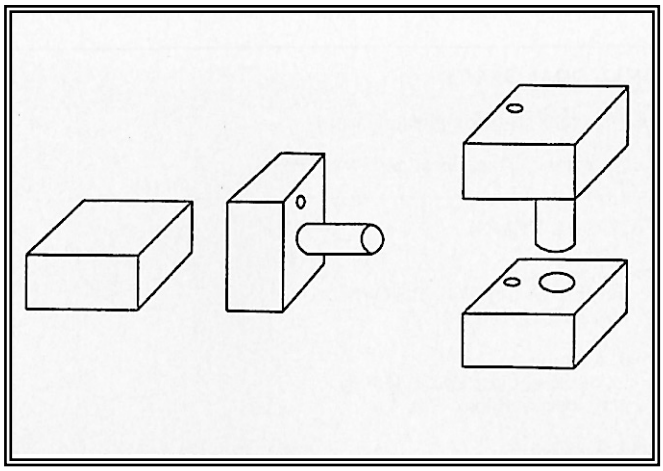


Figure 5. The Initial State and Assembly Goal for the Example.

To satisfy this goal, Spar examines its possible actions and selects the assemble action. Of course, the assemble action has both operational and geometric preconditions that must now be considered, so the planner pushes these onto the appropriate goal stacks. The goal stacks and plan action list are shown in figure 6.

At this point, a word about the meaning of the preconditions is in order. The assemble action has a precondition of the form

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

As we discussed in Constraint Manipulation, Spar associates a set of stable poses with each object. When two objects are to be mated, Spar imposes two constraints on the pose of the stationary object. First, none of the object's mating features can be obscured in this pose. Second, the mating vector must point into the object's friction cone. The set of stable poses that satisfies the first condition is easily determined by comparing each stable pose's set of occluded faces with the set of

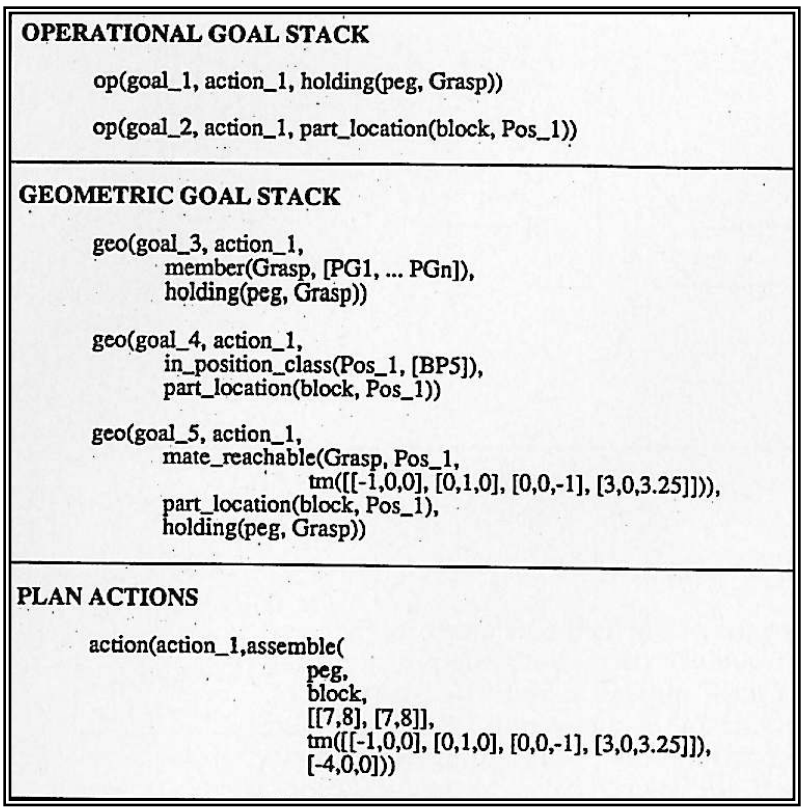


Figure 6. Goal Stacks and Plan Actions after the Addition of the Assemble Action.

mating features. The set of stable poses that satisfies the second condition is found by transforming the mating vector (which is expressed in the stationary object's coordinate system) into world coordinates and examining the *Z* component of the resulting vector. The set of permissible stable poses for the stable object is found as the intersection of these two sets. Note that for this example, there is only one stable pose of the block that satisfies both these conditions—the pose with the block resting on its back (this particular stable pose is denoted by the pose label BP5). Therefore, when the planner adds the assemble action to the plan, it instantiates the variable PositionList to the list [BP5]. (We should note that such a list of stable poses is actually a list of pose labels that are used to access 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 that is used to grasp the object but also the set of object features that are obscured by the grasp (as discussed in Reachability of Grasps). Therefore, it is a simple matter to determine which grasping configurations do not obscure the mating features of an object. When the planner adds the assemble action to the plan, it instantiates the variable GraspList to be this set of grasping configurations (because there are many of these, we do not explicitly list them here).

Figure 6 shows the instantiated versions of the preconditions for the assemble action, 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 specifies that the gripper be holding the peg in some valid grasp (remember that at the operational level, Spar is not concerned with the geometric aspects of the grasping configuration). Because it is not possible to merely add a constraint to the plan to achieve this goal (because 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 constraint 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 figure 7. Note that when Spar 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 these 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 that are labels for the homogeneous transformations representing the initial positions of the block and peg.) The top goal on the geometric goal stack, goal_8, is for the pickup action, and it specifies that the manipulator configuration used to pick up the peg, grasp_1, be physically realizable by the robot. To satisfy this goal, Spar attempts to add a constraint on the way in which grasp_1 is chosen so that the configuration will be reachable. Spar, therefore, instructs CMS to add the constraint reachable(grasp_1,init_pos2) to the constraint

network, as described in Reachability of Grasps. For our example, we assume that this constraint is consistent with the constraint network.

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 requirement of nonobscuration is expressed as a *member constraint,* that is, a restriction on the label set for the plan-variable grasp_1. Again, the planner invokes CMS to add the member constraint to the constraint network. Again, for the example, let us suppose that the new member constraint is consistent with the network. Note that if adding this constraint resulted in an inconsistent constraint set, Spar would be forced to insert additional manipulations.

To this point in the example, Spar has been able to satisfy geometric goals by merely adding constraints on operations already in the plan. In some cases, it is not possible to satisfy geometric goals this way, and an alternative approach must be used, as is the case for the geometric precondition goal_4, which constrains the possible positions of the block. As mentioned previously, there is only one permissible stable pose for the block, with the label BP5. When goal_2, part_location(block, Pos_1) is satisfied by the initial world state, the in_position_class goal fails (because the block is not initially resting in stable pose BP5). Spar cannot add a constraint on the block's initial position because it is a constant value that is defined by the initial world state. Furthermore, because no action currently in the plan manipulates the block, Spar cannot constrain the execution of a plan action to achieve the goal. Therefore, backtracking must be used to find some alternative method to satisfy goal_2.

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. Finding none, Spar adds the action putdown(block,pos_1). With this action added, 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, and the mating vector will point into the object's friction cone, which is the same as constraining pos_1 to be the block's stable pose BP5. In addition, Spar adds the constraint prior_to(action_3,action_1) to the ordering graph because the block must be placed in pos_1 prior to the assemble action. Of course, the addition of this action introduces new goals, so additional planning must be done. This planning, however, is very similar to the planning that must be done to pick up the peg appropriately, so we do not discuss it here.

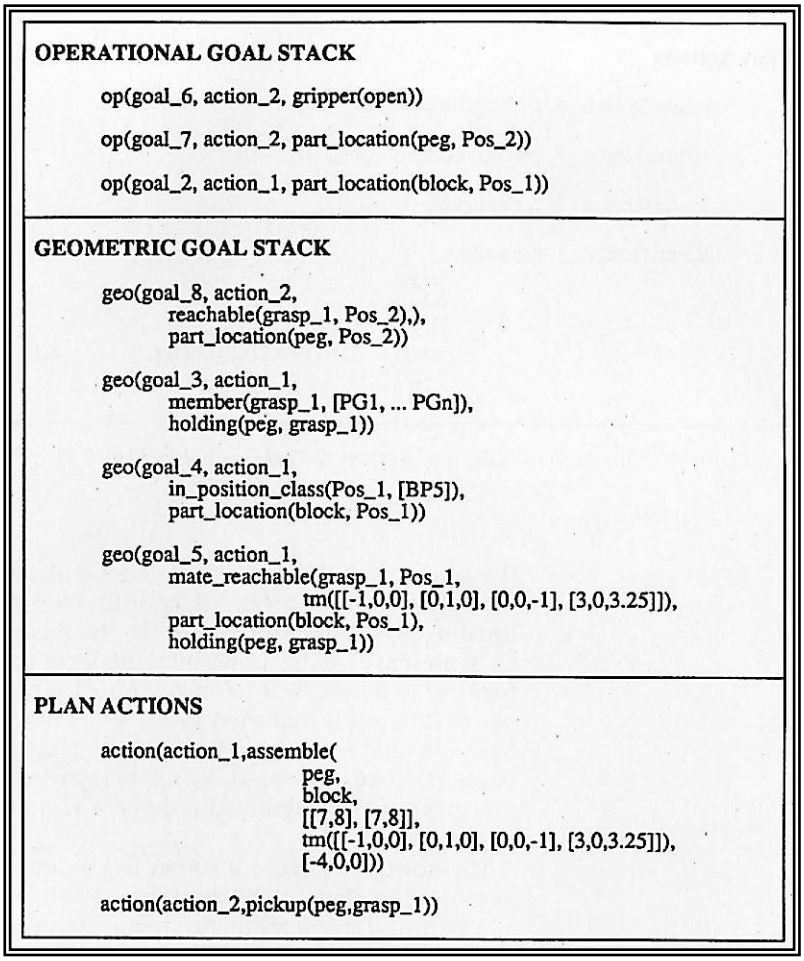


Figure 7. Goal Stacks and Plan Actions after the Addition of the Pickup Action.

The final result of the first phase of planning is shown in figures 8, 9, and 10. Figure 8 shows the four actions that are in the plan. The top of figure 9 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 place it on the table. Therefore, both init_pos_1 and pos_1 must be reachable using grasp_2, a requirement 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 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 figure 10, 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

```
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,
                                [[7,8], [7,8]],
                                tm([[-1,0,0], [0,1,0], [0,0,-1], [3,0,3.25]]),
                                [-4,0,0]))
```

Figure 8. Plan Actions Solve the Assembly Task Shown in Figure 5.

the label sets of the connected nodes and collecting those that meet the conditions outlined in Reachability of Grasps. In the figure, we represented stable poses by symbols of the form *xPy,* where *x* is used to indicate the object (the peg is indicated by $x = P$, the block by $x = B$), and *y* is used to indicate the specific stable pose for the object. Symbols representing grasping configurations have a similar interpretation.

The bottom of figure 9 shows the ordering graph. Note that in the ordering graph, in addition to the arcs we mentioned previously, there is an arc from action_3 to action_2. This arc is added to the graph because the action used to pick up the block (action_4) violates the gripper(open) operational goal for the action used to pick up the peg (action_2). To remedy this, action_3 is constrained to come between action_4 and action_2 to reestablish the gripper(open) goal.

Once the operational and geometric goals are satisfied, Spar considers the uncertainty-reduction goals. As we 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 only consider the uncertainty-reduction goals for the first pickup action, which were discussed in The Representation of Uncertainty in Spar.

To evaluate the constraints associated with these goals, Spar invokes the procedure that constructs and evaluates the relevant algebraic expressions for the goal. The resulting expression for the *Y* component of $P_1^{-1}C_1$ is

$$
\begin{aligned}
&-2.0 + 3.0*\cos(thetagr)*\cos(theta\_o) \\
&+ \cos(thetagr)*dx\_o + \sin(thetagr) \\
&+ \sin(thetagr)*dxgr + \\
&-1.0*\cos(thetagr)*\sin(theta\_o) + -1.5*\cos \\
&(thetagr) + -3.0*\sin(thetagr)*\sin(theta\_o) + \\
&-1.0*\cos(theta\_o)*\sin(thetagr) + \\
&-1.0*\sin(thetagr)*dy\_o + \\
&-1.0*\cos(thetagr)*dygr
\end{aligned}
$$

Note that *thetagr*, *dxgr*, *dygr*, and *dzgr* represent the uncertainties in the gripper configuration and that *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 Satisfying Uncertainty-Reduction Goals 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 omit these here. Using the SUP and INF routines with the bounds on the uncertainties listed in table 2, we find the lower bound on this expression to be 3.2793, which indicates that the constraint was satisfied. The remaining three constraints are similarly evaluated.

If the uncertainty-reduction goals are not satisfied in the world description, Spar attempts to add a sensing operation to the plan. Because it is impossible to predict the results of a sensing action, the add-delete lists for sensing actions merely describe the uncertainty in the object's location after the application of the sensing operation. If this reduction is sufficient, the sensing 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. Examples of sensing verification and local recovery plan templates are given in figures 11 and 12. The verification template in figure 11 is used to test the width of the gripper opening to ensure that the object was successfully grasped. If the gripper is not

Bounds on Uncertainty Variables		
Variable	Lower Bound	Upper Bound
dxgr	–0.001	0.001
dxgr	–0.001	0.001
dxgr	–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

Table 2. Bounds on Uncertainty Variables Used in the Example in Constraint Manipulation.

opened to the correct width, an error is signaled. The recovery template of figure 12 is used to respond to the possible errors. If error1 is signaled (that is, no object is grasped), then a local groping strategy is used to find the part. If error2 is signaled, the user is summoned.

Experimental Results

We have used Spar to plan a number of assembly tasks. The resulting assembly plans were then executed in a robot work cell equipped with a Puma 762 robot. Each of these tasks involved assembling the two objects described in A Task Planning Example. By varying the initial positions of the objects, Spar was forced to develop distinct plans for the individual tests, even though the assembly goals were the same.

Figures 13, 14, 15, and 16 illustrate a number of the experiments that have been performed. Each figure includes a photograph of the initial world situation and a listing of the plan that is output by Spar. For example, in figure 13, the block is initially face down, and the peg is resting on its side. To perform the assembly, the robot must first reposition the block so that the face containing the hole is accessible. This assembly requires two manipulations: one to place the block on its side (an intermediate position) and one to then place the block on its back. Once the block is repositioned, the peg is picked up, and the assemble action is performed. Figure 17 contains a sequence of photographs of the Puma 762 executing this plan. Note that in this sequence, the end points of each action are shown.

To have the robot perform the plans developed by Spar, each plan step is converted into a Lisp function call. The individual function calls that correspond to the actions in the plan are then collected into a list and written to a command file. This command file is used by the Lisp execution module that controls the robot. Table 3 describes the Lisp functions that are used to perform each of the actions used by Spar. Table 4 describes some of the Lisp functions that make up the interface to the robot. Note that the order of the actions shown in the lists in figures 13, 14, 15, and 16 reflects the order in which the actions were added to the plan. The index associated with an action (the first item in the sublist) indicates the order of action execution.

Because in its current implementation, Spar does not include a motion-planning module, the Lisp functions that execute the actions use a simple strategy to avoid collisions

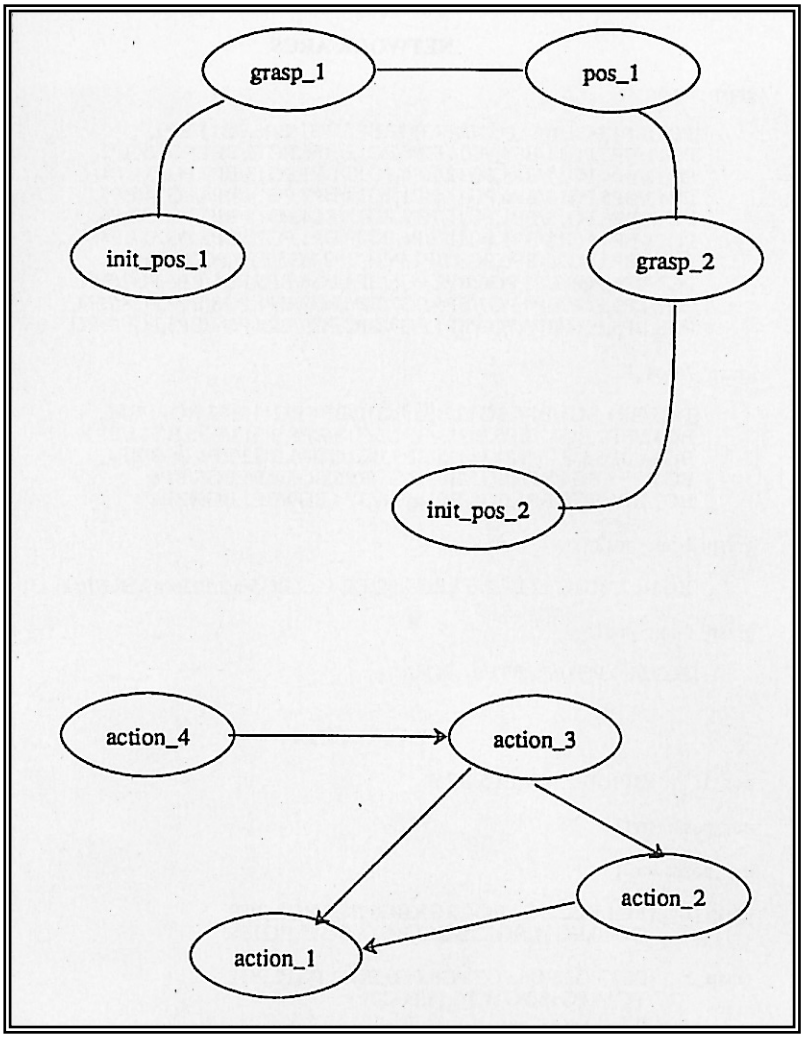


Figure 9. Constraint Network for the Assembly Plan to Solve the Task Shown in Figure 5.

Function	Arguments
(pickup-object obj tr**)**	obj: object to be grasped tr: transformation from object coordinate frame to manipulator coordinate frame
(putdown-object object tr**)**	obj: object to be placed on table tr: transformation from world coordinate frame to object coordinate frame
(assemble-objects obj1 obj2 tr vec**)**	obj1: held object obj2: object resting on table tr: transformation from resting object's coordinate frame to held object's coordinate frame vec: approach vector for mating operation

Table 3. Lisp Functions to Execute Spar Actions.

```
NETWORK ARCS

grasp_1-pos_1:

    [PG1/BP1,PG1/BP2,PG1/BP4,PG1/BP5,PG1/BP6,PG11/BP1,
     PG11/BP2,PG11/BP4,PG11/BP5,PG11/BP6,PG12/BP1,PG12/BP2,
     PG12/BP4,PG12/BP5,PG12/BP6,PG13/BP1,PG13/BP2,PG13/BP4,
     PG13/BP5,PG13/BP6,PG14/BP1,PG14/BP2,PG14/BP4,PG14/BP5,
     PG14/BP6,PG15/BP1,PG15/BP2,PG15/BP4,PG15/BP5,PG15/BP6,
     PG18/BP4,PG18/BP5,PG18/BP6,PG20/BP1,PG20/BP2,PG20/BP4,
     PG20/BP5,PG20/BP6,PG3/BP1,PG3/BP2,PG3/BP4,PG3/BP5,
     PG3/BP6,PG6/BP1,PG6/BP2,PG6/BP4,PG6/BP5,PG6/BP6,PG7/BP1,
     PG7/BP2,PG7/BP4,PG7/BP5,PG7/BP6,PG8/BP1,PG8/BP2,PG8/BP4,
     PG8/BP5,PG8/BP6,PG9/BP1,PG9/BP2,PG9/BP4,PG9/BP5,PG9/BP6]

grasp_2-pos_1:

    [BG1/BP1,BG1/BP4,BG10/BP2,BG10/BP4,BG11/BP2,BG11/BP6,
     BG12/BP2,BG12/BP5,BG13/BP1,BG13/BP5,BG13/BP6,BG14/BP1,
     BG14/BP6,BG15/BP1,BG15/BP5,BG16/BP5,BG2/BP1,BG3/BP4,
     BG4/BP5,BG4/BP6,BG5/BP5,BG5/BP6,BG6/BP5,BG6/BP6,
     BG7/BP6,BG8/BP1,BG8/BP2,BG8/BP4,BG9/BP1,BG9/BP2])

grasp_2-init_pos2:

    [BG1/tr2,BG16/tr2,BG2/tr2,BG3/tr2,BG4/tr2,BG5/tr2,BG6/tr2,BG7/tr2]

grasp_1-init_pos1:

    [PG12/tr1,PG3/tr1,PG7/tr1,PG8/tr1]

LABEL SETS

pos_1:      [BP1,BP2,BP4,BP5,BP6]

init_pos1:  [tr1]

init_pos2:  [tr2]

grasp_2:    [BG1,BG2,BG3,BG4,BG5,BG6,BG7,BG8,BG9,
             BG10,BG11,BG12,BG13,BG14,BG15,BG16]

grasp_1:    [PG1,PG3,PG6,PG7,PG8,PG9,PG11,PG12,PG13,
             PG14,PG15,PG17,PG18,PG20]
```

Figure 10. The Arcs and Label Sets for the Constraint Network Shown in Figure 9.

during the assembly process. The basic strategy is to move the end effector to a position above its destination and then use straight-line motion (in Euclidean space) to move to the destination. For example, the Lisp function for the pickup action is shown in figure 18. This function first moves the end effector to a position above the object to be grasped and then uses straight-line motion to move to the grasping position. The gripper fingers are then closed, and the sequence is reversed to raise the object to a position above the table. After the action is executed, the execution module's world model is updated to reflect the changes affected by the action.

Finally, we should note that although the uncertainty-reduction planner has been implemented, it has not yet been integrated with the execution system for two reasons. First, we have not yet fully and accurately

```
action-id:      ActionId,

parameter-list:

        WG = width of object
        W = width of gripper opening

error-states:

        W = WG → success

        W = 0 → error1

        true → error2
```

Figure 11. Template for a Sensory Verification Action.

```
action-id:      ActionId,

error-states:

        success → return true

        error1 → local-grope()

        error2 → summon-user()
```

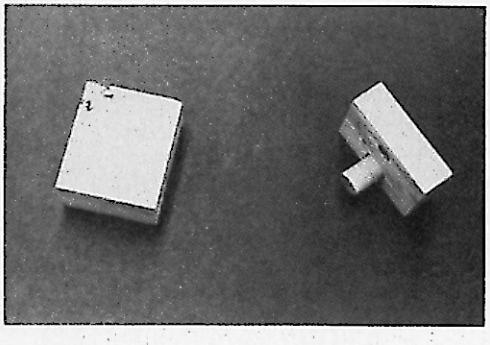
Figure 12. Template for a Local Recovery Plan.

characterized the uncertainties in the sensors and robotic manipulator. Second, the current set of recovery plans lacks robustness and generality. These two areas are the subjects of ongoing research.

Conclusions

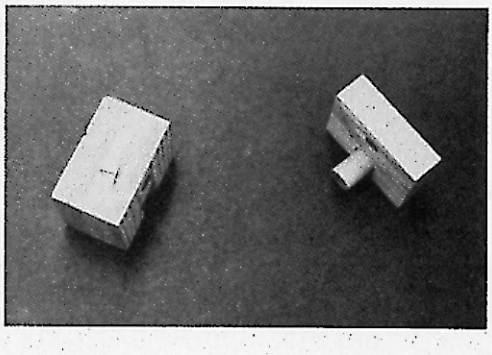
This article describes the move toward a planning system that can create assembly plans given as input a high-level description of assembly goals, geometric models of the assembly components, and a description of the capabilities of the work cell (including the robot and the sensory system). The resulting planner, Spar, reasons at three levels of abstraction: operational (where high-level operations are planned), geometric (where geometric configurations of the actions are planned), and uncertainty-reduction (where world uncertainties are taken into account).

At the first two levels of planning, we extended the constraint-posting approach used to date in domain-independent planning by adding geometric preconditions to the actions, linking these to operational goals using plan variables, and expanding CMS to



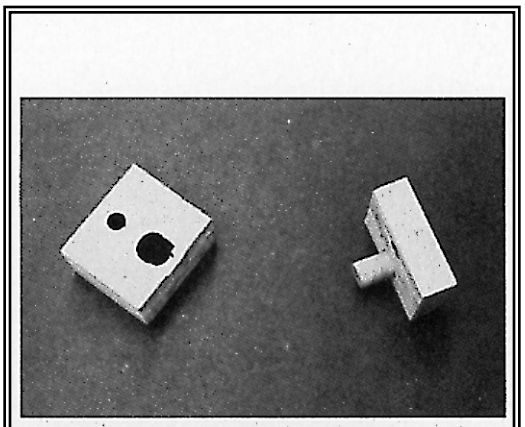
```
((1 (pickup-object block
          ((0.707107 0.0 -0.707107)
           (-0.0 1.0 0.0)
           (0.707107 0.0 0.707107)
           (1.0 1.375 1.625))))
 (2 (putdown-object block
          ((-1 0 0)
           (0 1 0)
           (0 0 -1)
           (-10 30 -18.05))))
 (3 (pickup-object block
          ((-0.707107 8.6593e-17 -0.707107)
           (-1.2246e-16 -1.0 0.0)
           (-0.707107 8.6593e-17 0.707107)
           (1.0 1.375 1.625))))
 (4 (putdown-object block
          ((0 0 1)
           (0 1 0)
           (-1 0 0)
           (-10 20 -21.3))))
 (5 (pickup-object peg
          ((-0.707107 -0.707107 -6.123e-17)
           (8.6593e-17 6.163e-33 -1.0)
           (0.707107 -0.707107 6.123e-17)
           (0.5 1.375 1.625))))
 (6 (assemble-objects peg block
          ((-1 0 0)
           (0 1 0)
           (0 0 -1)
           (3 0 3.25))
          (-4 0 0))))
```

Figure 13. Plan to Assemble Two Objects with the Block Face Down in the Initial State.



```
((1 (pickup-object block
          ((0.707107 -0.707107 -6.123e-17)
           (8.6593e-17 6.163e-33 1.0)
           (-0.707107 -0.707107 6.123e-17)
           (1.0 1.375 1.625))))
 (2 (putdown-object block
          ((0 0 1)
           (0 1 0)
           (-1 0 0)
           (-10 30 -21.3))))
 (3 (pickup-object peg
          ((-0.707107 -0.707107 -6.123e-17)
           (8.6593e-17 6.163e-33 -1.0)
           (0.707107 -0.707107 6.123e-17)
           (0.5 1.375 1.625))))
 (4 (assemble-objects peg block
          ((-1 0 0)
           (0 1 0)
           (0 0 -1)
           (3 0 3.25))
          (-4 0 0))))
```

Figure 14. Plan to Assemble the Two Objects with the Block and the Peg Initially on Their Sides.

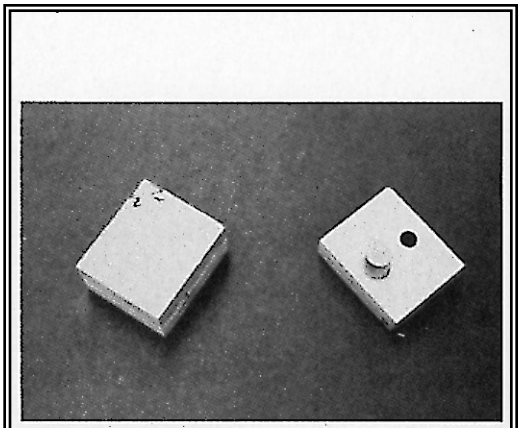


```
((1 (pickup-object peg
          ((-0.707107 -0.707107 -6.123e-17)
           (8.6593e-17 6.163e-33 -1.0)
           (0.707107 -0.707107 6.123e-17)
           (0.5 1.375 1.625))))
 (2 (assemble-objects peg block
          ((-1 0 0)
           (0 1 0)
           (0 0 -1)
           (3 0 3.25))
          (-4 0 0))))))
```

Figure 15. Plan to Assemble Two Objects with the Block Face Up in the Initial State.

deal with geometric constraints. At the uncertainty-reduction level of planning, we 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, a number of areas in our system are either ad hoc or require 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 work is to automate this process by employing geometric reasoning about possible errors and error recovery. Another shortcoming of Spar is the lack of a motion-planning module. Incorporating a motion planner into the current system is another goal.



```
((3 (pickup-object block
          ((0.707107 0.0 -0.707107)
           (-0.0 1.0 0.0)
           (0.707107 0.0 0.707107)
           (1.0 1.375 1.625))))
 (4 (putdown-object block
          ((-1 0 0)
           (0 1 0)
           (0 0 -1)
           (-10 30 -18.05))))
 (5 (pickup-object block
          ((-0.707107 8.6593e-17 -0.707107)
           (-1.2246e-16 -1.0 0.0)
           (-0.707107 8.6593e-17 0.707107)
           (1.0 1.375 1.625))))
 (6 (putdown-object block
          ((0 0 1)
           (0 1 0)
           (-1 0 0)
           (-10 20 -21.3))))
 (1 (pickup-object peg
          ((-0.707107 8.6593e-17 -0.707107)
           (-1.2246e-16 -1.0 0.0)
           (-0.707107 8.6593e-17 0.707107)
           (1.0 1.375 1.625))))
 (2 (putdown-object peg
          ((-1 0 0)
           (0 1 0)
           (0 0 -1)
           (-10 25 -21.3))))
 (7 (pickup-object peg
          ((0.707107 0.0 -0.707107)
           (-0.0 1.0 0.0)
           (0.707107 0.0 0.707107)
           (0.5 1.375 1.625))))
 (8 (assemble-objects peg block
          ((-1 0 0)
           (0 1 0)
           (0 0 -1)
           (3 0 3.25))
          (-4 0 0))))))
```

Figure 16. Plan to Assemble the Two Objects with the Block on Its Back and the Peg Face Up in the Initial State.

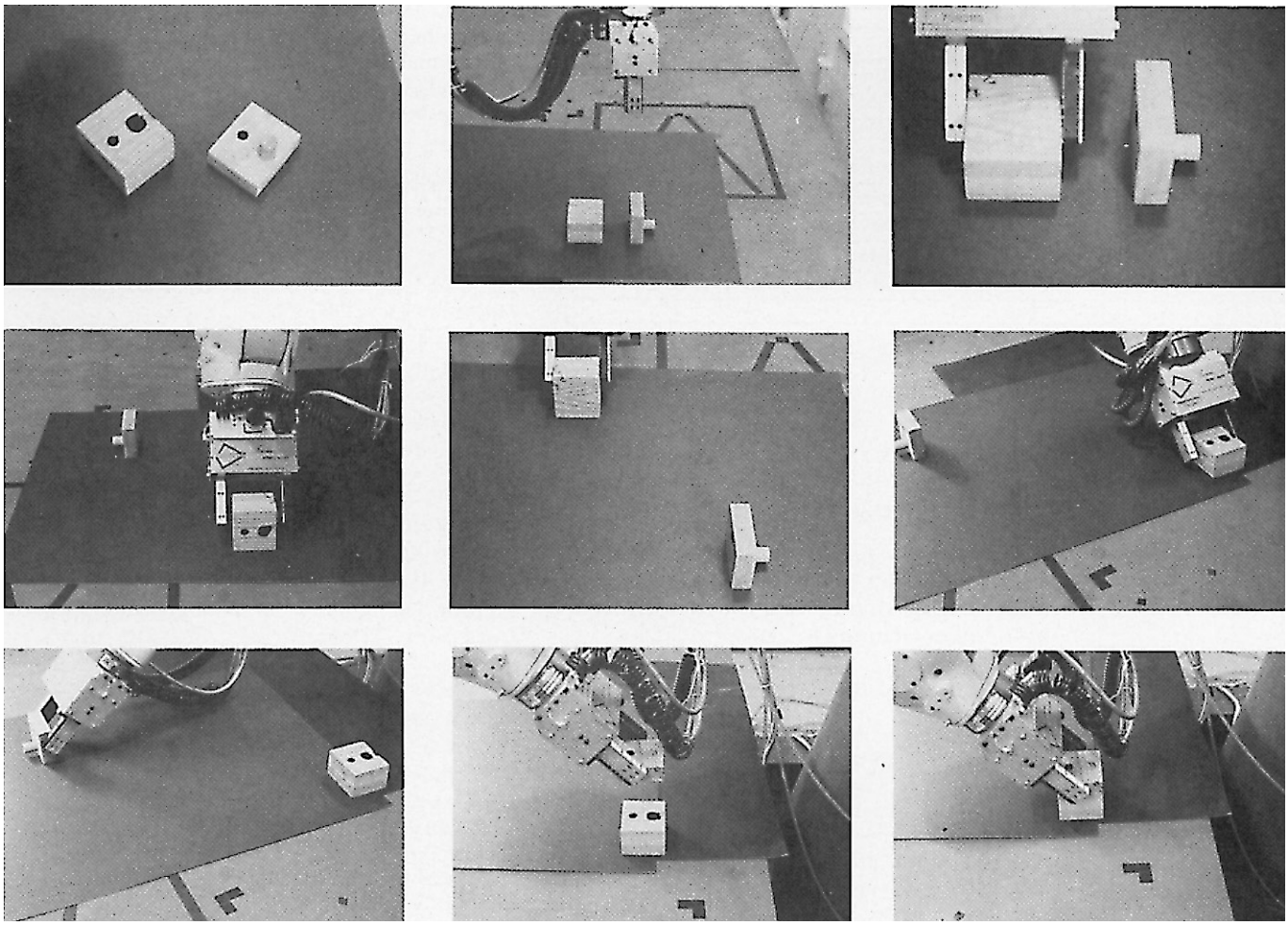


Figure 17. The Puma Robot Executing the Plan.

Figure 17a shows the two objects to be assembled. In the goal configuration, the peg will be inserted into the block so that the two small holes are aligned. Figure 17b shows the initial world state for the assembly plan shown in figure 13. Note that the block is face down and, therefore, must be inverted prior to assembling the two parts. As shown in figure 13, inverting the block requires two pickup-putdown action pairs, the first to place the block in an intermediate position, the second to place the block on its back. Figure 17c shows the robot grasping the block to begin the first pickup block action. Figure 17d shows the robot placing the block on the table in the first putdown block action. Note that the block is placed on its side so that it can be grasped and repositioned on its back. Figure 17e shows the robot grasping the block to begin the second pickup block action. Figure 17f shows the robot placing the block on the table in the second putdown block action. Note that the block is now resting on its back, and the two parts can now be assembled. Figure 17g shows the robot grasping the peg to begin the pickup peg action. Figure 17h shows the robot poised for the execution of the assemble-objects peg block action. Figure 17i shows the completion of the assemble-objects peg block action.

```
(defun pickup-object (object grasp-tr)

  (let ((object-position (get-object-position object))
        (absolute-position (transform-mult object-position grasp-tr)))

    (az *safe-approach-height*)
    (move-robot-at-safe-height absolute-position)

    (straight-motion)

    (gripper-open)
    (move-robot absolute-position)
    (gripper-close)

    (joint-motion)

    (move-robot-at-safe-height absolute-position )

    (put-current-grasp object grasp-tr)
    (put-object-position object nil)))
```

Figure 18. The Lisp Procedure to Execute the Pickup Action.

The function az *is used to move the manipulator to a specified height, move-robot moves the manipulator to the specified position (given as a homogeneous transformation matrix), and move-robot-at-safe-height moves the robot to a destination position and keeps the manipulator at a specified position above the work table.*

Function	Description
(az Z-position)	Move manipulator to the absolute position specified by Z-position
(gripper-open)	Open the fingers of the manipulator
(gripper-close)	Close the fingers of the manipulator
(move-robot TR)	Move the robot to the configuration indicated by the homogeneous transformation TR
(move-robot-at-safe-height TR)	Put the robot into the configuration specified by TR, but at some predefined Z position

Table 4. Miscellaneous Lisp Functions to Control the Robot.

Acknowledgments

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

Notes

1. 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 such as threading and fixturing could be considered more specialized forms of the assemble action presented here, the specialized forms being obtained by adding more geometric and uncertainty-reduction constraints.

2. An introductory treatment of homogeneous transformation matrixes can be found in Paul (1981).

References

Ambler, A. P., and Popplestone, R. J. 1975. Inferring the Positions of Bodies from Specified Spatial Relationships. *Artificial Intelligence* 6:157–174.

Bledsoe, W. W. 1975. The SUP-INF Method in Presburger Arithmetic, Memo ATP-18, Mathematics Dept., Univ. of Texas at Austin.

Boyer, M., and Daneshmend, L. K. 1987. An Expert System for Robot Error Recovery, Technical Report, TR-CIM-87-18, Computer Vision and Robotics Laboratory, McGill University.

Brooks, R. A. 1982. Symbolic Error Analysis and Robot Planning. *The International Journal of Robotics Research* 1(4): 29–68.

Brooks, R. A. 1981. Symbolic Reasoning among 3D Models and 2D Images. *Artificial Intelligence* 17:285–348.

Brost, R. C. 1985. Planning Robot Grasping Motions in the Presence of Uncertainty, Technical Report, CMU-RI-TR-85-12, Computer Science Dept., Carnegie-Mellon Univ.

Chapman, D. 1987. Planning for Conjunctive Goals. *Artificial Intelligence* 32(3): 333–378.

Chen, C. H., and Kak, A. C. 1989. A Robot Vision System for Recognizing 3-D Objects in Low-Order Polynomial Time. *IEEE Transactions on Systems, Man, and Cybernetics* 19(6): 1535–1563.

Davis, E. 1987. Constraint Propagation with Interval Labels. *Artificial Intelligence* 32(3): 281–331.

Dechter, R., and Pearl, J. 1987. Network-Based Heuristics for Constraint-Satisfaction Problems. *Artificial Intelligence* 34(1): 1–38.

Donald, B. R. 1987. A Search Algorithm for Motion Planning with Six Degrees of Freedom. *Artificial Intelligence* 31(3): 295–353.

Donald, B. R. 1986. Robot Motion Planning with Uncertainty in the Geometric Models of the Robot and Environment: A Formal Framework for Error Detection and Recovery. In Proceedings of the IEEE International Conference on Robotics and Automation, 1588–1593. Washington, D.C.: Computer Society Press.

Durrant-Whyte, H. F. 1988. Uncertain Geometry in Robotics. *IEEE Journal of Robotics and Automation* 4(1): 23–31.

Erdmann, M. A. 1984. On Motion Planning with Uncertainty, Technical Report, AI-TR-810, Artificial Intelligence Lab., Massachusetts Institute of Technology.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2:189–208.

Gini, M.; Doshi, R.; Gluch, M.; Smith, R.; and Zualkernan, I. 1985. The Role of Knowledge in the Architecture of a Robust Robot Control. In Proceedings of the IEEE International Conference on Robotics and Automation, 561–567. Washington, D.C.: Computer Society Press.

Gottschlich, S. N., and Kak, A. C. 1989. A Dynamic Approach to High Precision Parts Mating. *IEEE Transactions on Systems, Man, and Cybernetics* 19(4): 797–810.

Hoffman, R. 1989. Automated Assembly in a CSG Domain. In Proceedings of the IEEE International Conference on Robotics and Automation, 210–215. Washington, D.C.: Computer Society Press.

Homem de Mello, L. S., and Sanderson, A. C. 1989. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. In Proceedings of the IEEE International Conference on Robotics and Automation, 56–61. Washington, D.C.: Computer Society Press.

Huang, Y. F., and Lee, C. S. G. 1989. Precedence Knowledge in Feature Mating Operation Assembly Planning. In Proceedings of the IEEE International Conference on Robotics and Automation, 216–221. Washington, D.C.: Computer Society Press.

Hutchinson, S. A., and Kak, A. C. 1989. Planning Sensing Strategies in a Robot Work Cell with Multi-Sensor Capabilities. *IEEE Transactions on Robotics and Automation* 5(6): 765–783.

Hutchinson, S. A., and Kak, A. C. 1988. A Task Planner for Simultaneous Fulfillment of Operational, Geometric, and Uncertainty-Reduction Goals. In Proceedings of the Symposium on

Human-Machine Symbiotic Systems, Oak Ridge National Lab 37-77. Also a more detailed account in Technical Report, TR-EE 88-46, Purdue Univ.

Hwang, Y. K., and Ahuja, N. 1988. Path Planning Using a Potential Field Representation, Technical Report, UICU-ENG-88-2251, Univ. of Illinois at Urbana-Champaign.

Lee, C. S. G., and Ziegler, M. 1983. A Geometric Approach in Solving the Inverse Kinematics of PUMA Robots. In Proceedings of the Thirteenth International Symposium on Industrial Robotics and Robots 7, 16-1–16-18. Chicago: Society of Manufacturing Engineers.

Liu, Y. 1987. A Sensor-Based Robot Planner for Assembly Tasks. In Proceedings of the First International Artificial Intelligence Conference in Japan, 571–584. Osaka.

Liu, Y., and Popplestone, R. J. 1989. Planning for Assembly from Solid Models. In Proceedings of the IEEE International Conference on Robotics and Automation, 222–227. Washington, D.C.: Computer Society Press.

Lozano-Perez, T. 1987. A Simple Motion-Planning Algorithm for General Robot Manipulators. *The IEEE Journal of Robotics and Automation* RA-3(3): 224–239.

Lozano-Perez, T.; Jones, J. L.; Mazer, E.; O'Donnell, P. A.; Grimson, W. E. L.; Tournassound, P.; and Lanusse, A. 1987. Handey: A Robot System That Recognizes, Plans, and Manipulates. In Proceedings of the IEEE International Conference on Robotics and Automation, 843–849. Washington, D.C.: Computer Society Press.

Lozano-Perez, T., and Brooks, R. A. 1985. An Approach to Automatic Robot Programming, Technical Report, AIM 842, Massachusetts Institute of Technology.

Lozano-Perez, T.; Mason, M. T.; and Taylor, R. H. 1984. Automatic Synthesis of Fine-Motion Strategies for Robots. *The International Journal of Robotics Research* 3(1): 3–24.

Nof, S. Y.; Maimon, O. Z.; and Wilhelm, R. G. 1987. Experiments for Planning Error-Recovery Programs in Robotic Work, Research Memo, 87-2, School of Industrial Engineering, Purdue Univ.

Paul, R. P. 1981. *Robot Manipulators: Mathematics, Programming, and Control*. Cambridge, Mass.: MIT Press.

Pertin-Troccaz, J. 1987. On-Line Automatic Robot Programming: A Case Study in Grasping. In Proceedings of the IEEE International Conference on Robotics and Automation, 1292–1297. Washington, D.C.: Computer Society Press.

Pertin-Troccaz, J., and Puget, J. 1987. Dealing with Uncertainties in Robot Planning Using Program Proving Techniques. In Proceedings of the Fourth International Symposium of Robotic Research.

Popplestone, R. J.; Ambler, A. P.; and Bellos, I. M. 1978. A Language for Describing Assemblies. In *The Industrial Robot,* 131–137.

Sacerdoti, E. D. 1977. A Structure for Plans and Behavior. New York: North-Holland.

Schwartz, J. T., and Sharir, M. 1988. A Survey of Motion Planning and Related Geometric Algorithms. *Artificial Intelligence* 37:157–169.

Shostak, R. E. 1977. On the SUP-INF Method for Proving Presburger Formulas. *Journal of the ACM* 24(4): 529–543.

Smith, R. C., and Cheeseman, P. 1986. On the Representation and Estimation of Spatial Uncertainty. *The International Journal of Robotics Research* 5(4): 56–68.

Smith, R. E., and Gini, M. 1986. Reliable Real-Time Robot Operation Employing Intelligent Forward Recovery. *Journal of Robotic Systems* 3(3): 281–300.

Srinivas, S. 1978. Error Recovery in Robots through Failure Reason Analysis. In Proceedings of the American Federation of Information Processing Societies National Computer Conference, 275–282. Washington, D.C.: American Federation of Information Processing Societies.

Stefik, M. 1981. Planning with Constraints (MOLGEN: Part 1). *Artificial Intelligence* 16:111–140.

Tournassound, P., and Lozano-Perez, T. 1987. Regrasping. In Proceedings of the IEEE International Conference on Robotics and Automation, 1924–1928. Washington, D.C.: Computer Society Press.

Wilkins, D. E. 1983. Representation in a Domain-Independent Planner. In Proceedings of the Eighth International Joint Conference on Artificial Intelligence, 733–740. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Avinash Kak is a professor of electrical engineering at Purdue University. His research interests are in reasoning architectures for solving spatial problems, sensor-based robotics, and computer vision. He has coauthored the books *Digital Picture Processing* (Academic Press) and *Principles of Computerized Tomographic Imaging* (IEEE Press). His address is Robot Vision Lab, EE Building, Purdue University, West Lafayette, IN 47907.

Seth Hutchinson was with the Robot Vision Lab at Purdue University during the course of the research reported here. He is currently an assistant professor in the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory, and the Beckman Institute at the University of Illinois, Urbana, Illinois. His current research interests include dynamic planning of sensing strategies, constraint-based reasoning, task planning in automated assembly, evidential reasoning applied to model-based object recognition, and sensor integration.