Planning and Reasoning in Sensor Based Robotics

A. C. Kak, S. A. Hutchinson and K. M. Andress

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

ABSTRACT

This paper surveys the planning and reasoning research being carried out in the Robot Vision Lab at Purdue. In particular, we will describe the workings of a new planning system called SPAR, which uses a constraint posting approach for simultaneously fulfilling the operational, geometric and uncertainty reduction goals, and the PSEIKI system for evidential reasoning in a tangled hierarchy. We will also mention briefly our other related research in high precision assembly under force/torque control and robotic manipulation with structural stereopsis for depth perception.

1. INTRODUCTION

The aim of this article is to survey the ongoing research in the areas of planning and reasoning at the Robot Vision Lab at Purdue. In each area, only the key concepts will be highlighted here, for more detailed information the reader will be referred to other publications.

Our interest in planning stems from our desire to give an assembly robot the capability to expand on its own a human supplied top-level assembly plan into a sequence of manipulations, accompanied by appropriate sensory invocations, to accomplish an assembly task, it being assumed that the parts to be assembled are in random positions and orientations. The sensory invocations serve two purposes: they help reduce uncertainties that may be associated with the identities and placements of the parts; additionally, they can be used to verify the successful termination of a particular sequence of manipulations. In addition to sensory invocations, some manipulation steps may also be used for uncertainty reduction; for example, the mere act of gripping an object with a two-fingered end-effector reduces the uncertainty in the location of the object along the direction joining the two fingers.

Much of the research work done to date on planning is characterized by unrealistic idealizations about the domain objects, the environment, and the actions that can be executed towards the fulfillment of the goals. The domain objects are perceived as symbolic entities whose behavior is entirely predictable with respect to the actions. Each action is modeled as a deterministic operation whose consequences are predictable, too. In most of these planning procedures, the state of the domain can be modeled as sentences from a suitable language, and each action as a transformation that maps a sentence into another sentence. As was eloquently pointed out in [15], such planning, useful only as *offline strategic planning*, serves purpose more as a theoretical exercise for studying important issues like the frame problem, how to focus search, goal representation, etc., than as a tool for actually planning, say, assembly robots. Idealizations incorporated in this prior work on planning are tantamount to divorcing from consideration execution time issues like our lack of knowledge of the precise location of objects in a robot work cell, uncertainty (representable by an appropriate formalism using some form of confidence numbers) about the identities of the various objects, etc., and the need for the invocation of the available sensors and manipulations for reducing uncertainties to a level that would keep the robot from "freezing" or "thrashing" in its attempts to reach a goal.

In Section 2, we will describe our approach to task level planning that was inspired by Chapman's work on the constraint posting method. [A constraint may be viewed as a specification or a restriction on an action.] In the constraint posting method, one seeks to satisfy a goal by first examining all the actions and the constraints previously generated to see if the new goal can be satisfied by the addition of a new constraint. Only if this strategy fails, is a new action added. Chapman's planner by itself would be incapable of handling the geometric and uncertainty caused complexities in a robot work cell, and its main virtue lies in the fact that it possesses some elegant theoretical properties, such as the property of completeness which implies that if a solution to a planning problem exists the planner would find it.

Our task planner, described in greater detail in [9], expands upon Chapman's constraint posting idea by the inclusion of geometric and uncertainty goals; such goals are unavoidable in a robot work cell since in many assembly operations geometric constraints must be satisfied during the process of parts mating, and then there is, of course, the ever present uncertainty about the location and orientation of parts that has to be dealt with. In Section 2, we will briefly outline the reasoning architecture on which our planner is based.

We will then separately in Section 3 address that component of planning that focuses on the selection of an optimum sensor in a multi-sensor environment. Initial reports on this work appeared in [7,8]. Here we will only highlight the main concept of sensor selection on the basis of hypothesis disambiguation by the use of aspect graphs of objects.

In Section 4, we will take up the subject of reasoning over sensory information and briefly review our work on PSEIKI, a production system for integrating knowledge with images. Although expected to be useful for verification vision for assembly robots, currently PSEIKI's main application is for a near-sighted robot navigating over a network of roads. Also, PSEIKI serves as a testbed for experimenting with hierarchical evidential reasoning. Although, currently, the Dempster-Shafer theory is used for the purpose, we see no reason why one could also not use Bayesian based formalisms, especially using the Bayes-net methodology developed by Pearl [14]. Hierarchical reasoning owes its importance to the general belief among researchers that we humans invoke knowledge and constraints in hierarchies for the purpose of disambiguating hypotheses about the nature and identity of sensed data. In addition, there is the issue of computational complexity, which becomes more tractable when the sensed information and the constraints to be invoked for the interpretation thereof are all organized along semantically relevant hierarchies. By semantic relevance, we mean that the groupings organized at different scales in the sensed data should lend themselves to the imposition of geometric and topological constraints, also organized along hierarchies.

PSEIKI generates initial labels for the elements of the sensed data by a direct geometric comparison of the data with the model information. Based on the strengths of these comparisons, belief values are also assigned to these initial labels. Subsequently, the belief values are revised on the basis of the consistency of the assigned labels to neighboring data elements. Also, PSEIKI seeks to form groupings in the sensed data by paralleling the groupings in the model information. The initial labels assigned to data elements may be discarded if these labels result in higher level groupings that are in conflict with the labels of other groupings.

Finally, in Section 5 we have pointed to our other contributions, mostly in the areas of sensor driven manipulation and depth perception, that will have a bearing on the intelligent work cells of the future. Our work on force/torque sensing, which incorporates automatic error detection and recovery for peg-in-hole type of assembly experiments, now allows us to accomplish parts mating at tolerance levels of less than 0.001". Our work on structural stereopsis has led to manipulation experiments with rod like objects where depth perception was generated by the enforcement of relational constraints in a scene. Our work on the integration of CAD with manipulation was a consequence of the realization that an intermediate representation was needed to connect the CAD representation on the one hand and the representations used for vision processing. Our more recent work on rule based stereopsis has allowed us to integrate in an opportunistic framework the various complimentary strategies that have been developed to date for binocular fusion.

2. SPAR: A PLANNER FOR SIMULTANEOUS FULFILLMENT OF OPERATIONAL, GEOMETRIC AND UNCERTAINTY GOALS

Recently, Chapman has presented a formal algorithm for domain independent planning based on constraint posting [4]. His system, TWEAK, performs nonlinear planning at a single level of abstraction. Basically, TWEAK continually refines partial plans, attempting to ensure that each goal will be satisfied at the appropriate phase of plan execution. In order to ensure that a goal, g, is satisfied in a particular world state, s, the planner must ensure that there is a step in the plan which establishes g and that no step in the plan clobbers (undoes the goal) g before state s is reached. Establishing a goal amounts to finding an existing step t (or inserting one) in the plan which establishes g and constraining t to occur before s. As for dealing with the clobberers, for each action, C, that could clobber g, the planner must ensure that C occurs after s, or that there is some step w (or one can be added to the plan) such that C occurs before w, w occurs before s, and whenever C clobbers g then w reestablishes g (thus the term "white knight" to describe such w's).

Although TWEAK is ideal for constructing high level task plans, it isn't capable of the type of planning required to construct complete robot manipulation plans. The reason for this is that the representations TWEAK uses are not nearly powerful enough to allow for planning geometric configurations or for taking the uncertainties in the environment into consideration. This is because of the limitations that the plans treat the domain objects as symbolic entities, and that the effects of actions cannot be influenced by the context in which those actions are applied. Clearly these limitations are too severe for even simple robot task planning. For example, the act of parts mating will certainly be affected by the way in which the manipulator grasps the object to be mated, and is therefore context dependent. Furthermore, in order to deal with the uncertainties in the environment, we must increase the scope of the planning to include the geometric entities in the world, since the uncertainties in the environment are most often manifested as errors in quantities which describe the world's geometry, for example the locations and orientations of parts in the work cell.

Even though this domain independent type of planning cannot create complete robot task plans, it can be useful as a top level planner. In particular, we can use this type of planning to derive a preliminary set of robot operations which will be required to perform the desired task, for example, pick up a peg, insert the peg into a hole, etc. We refer to this level of planning as *operational planning*, and use it as the first step in constructing robot manipulation plans. Our approach to operational level planning is not significantly different from the approach described by Chapman, so we will not go into the details here.

In order to extend our planning to the geometric and uncertainty levels, we must extend the representation used for the planner's actions to include both geometric and uncertainty preconditions. We must also devise some way to link the operational level plans to the geometric and uncertainty levels of planning. For this purpose, in SPAR, which stands for Simultaneous Planner for Assembly Robot, at the operational level of planning, plan variables are introduced which can be constrained by the other levels of planning in order to determine how an action is executed. The geometric and uncertainty preconditions are expressed in terms of those variables. For example, a traditional STRIPS type action is (pickup ?Object). SPAR's equivalent action is (pickup ?Object ?Grasp). The variable ?Grasp is used to define the geometric configuration which will be used by the manipulator in grasping the object. At the operational level, ?Grasp is primarily ignored, but its presence gives that part of SPAR which handles geometric goals a method of constraining how the pickup operation is actually performed, thus linking distinct levels of planning.

In SPAR the preconditions for the geometric and uncertainty levels are not simple statements (as is the case at the operational level). Instead, each has two main components. The first is a geometric/uncertainty precondition and the second is an operational goal. The meaning of this pair is that the planner is to establish the operational goal in such a way that the geometric/uncertainty goal is satisfied. For example, one geometric precondition of the putdown action shown in Fig. 1 is:

```
(?C2 ?Step-id
     (reachable ?Grasp ?Position)
     (holding ?Object ?Grasp))
```

```
(?step-id
   (pickup ?Object ?Grasp)
   (Preconditions
      (op)
      (geo   (?C5 ?step-id
                (reachable ?Grasp ?Position)
                (part-location ?Object ?Position)))
      (unc   (?C6 ?step-id
                (<= dx (- GRIPPER-WIDTH ?object-width))
                (part-location-error ?Object ?dx ?dy ?dz ?dr))))
   (Add-list (holding ?Object ?Grasp))
   (Delete-list (part-location ?Object ?Position)
                (part-location-error ?Object . ?OLD-ERROR)))
(?step-id
   (putdown ?Object ?Position)
   (Preconditions
      (op    (?C1 ?step-id (holding ?Object ?Grasp)))
      (geo   (?C2 ?step-id
                (reachable ?Grasp ?Position)
                (holding ?Object ?Grasp)))
      (unc))
   (Add-list (part-location ?Object ?Position))
   (Delete-list (holding ?Object ?Grasp)))
(?step-id
   (assemble ?Obj1 ?Obj2 ?M-faces ?Transform ?M•vector)
   (Preconditions
      (op    (?C1 ?step-id (holding ?Obj1 ?Grasp)))
      (geo   (?C2 ?step-id
                (member ?Grasp ?Grasp-list)
                (holding ?Obj1 ?Grasp))
             (?C3 ?step-id
                (in-position-class ?Pos2 ?Position-list)
                (part-location ?Obj2 ?Pos2)))
      (unc))
   (Add-list (assembled ?Obj1 ?Obj2))
   (Delete-list (holding ?Obj1 ?Grasp)))
(?step-id
   (range-scan ?Object)
   (Preconditions
      (op    (?C1 ?step-id (gripper-state open))
             (?C2 ?step-id (part-location ?Object . ?Location))
      (geo)
      (unc))
   (Add-list (part-location-error ?Object 0.5 0.5 0.5 5.0))
   (Delete-list (part-location-error ?Object . ?Old-error)))
```

Fig. 1: A few of SPAR's actions.

In order to satisfy this condition, the planner searches for an action in the plan which achieves the goal (holding ?Object ?Grasp) and then attempts to constrain the execution of that action so that the condition (reachable ?Grasp ?Position) is met. In order to determine whether or not ?Position is reachable, the constraint manipulation system is invoked, which in turn invokes the kinematic routines for the robot. The variables ?C2 and ?Step-id are not important in this discussion. They are instantiated by the planner when the putdown action is added to the plan. The variable ?C2 is instantiated to a label that the planner uses to refer to this specific precondition, and the variable ?Step-id is instantiated to the label which the planner assigns to the putdown action.

The plans developed by SPAR are not simple, linear sequences of actions. Instead, plans consist of an unordered set of actions and a separate set of constraints on how and when those actions are to be executed. Therefore, a plan developed by SPAR actually corresponds to a family of plans. Any specific plan which can be derived by choosing values for the plan variables so that the constraints in the constraint database are satisfied constitutes an acceptable task plan. The process of planning, then, consists of adding actions and/or constraints to the plan (i.e. to the plan's action list or constraint database) until the system's goals are satisfied by the plan.

The planner must also keep track of its progress in solving goals. To do this, SPAR maintains three pending goal stacks (one each for operational, geometric and uncertainty goals) and a list of goals which have already been satisfied by the system. Initially, the pending goal stacks contain only those goals which are entered by a human user, and the list of satisfied goals is empty. During the planning process, any time an action is added to the plan, its preconditions are added to the appropriate pending goal stacks. The reason for the list of satisfied goals is that the introduction of a new action into the plan could possibly undo a previously satisfied goal. Therefore, whenever a new action is introduced into the plan, the planner checks the satisfied goals, and any which are undone by the new action are moved to the list of pending goals.

A great deal of the actual work done during planning is related to the addition of constraints to the plan. The planner cannot blindly add a new constraint to the plan's constraint set. It must first make sure that the new constraint is not inconsistent with the current constraint set. For example, in order to satisfy the goal that an object is in a certain position on the work table, the planner cannot simply add such a constraint to the plan. If the object is already in some position which does not satisfy the constraint, then this new constraint would cause an inconsistency. Determining whether or not a new constraint is consistent with the current constraint set is done by the constraint manipulation system (CMS). If a new constraint is inconsistent with the current constraint set, then the CMS signals a failure to the planner so that other options can be explored. Otherwise, the CMS derives the new constraints on plan variables which are implied by the new constraint. Fig. 2 lists some of the constraints used in our system, along with their meaning.

Constraints

(prior-to *?step-1 ?step-2*)
    ?step-1 must be executed prior to ?step-2

(reachable *?grasp ?position*)
    ?grasp must be chosen so that ?position is reachable

(member *?item ?list*)
    ?item must be chosen from ?list

(in-position-class *?position ?position-list*)
    ?position must be an element of ?position-list

($\leq$ *?expr-1 ?expr-2*)
    ?expr-1 must be $\leq$ to ?expr-2

Fig. 2: Some of SPAR's constraints.

The various components of the planner, and the data structures that it maintains, are shown in the block diagram in Fig. 3. Note that the planner does not directly interact with the constraint database. The planner can only request the addition of a constraint. The CMS determines whether or not that constraint is consistent with the current constraint set.

There are two basic methods in SPAR for satisfying goals. The planner can add a new action to the plan, or it can constrain the execution of some action which is already in the plan so that it will achieve the goal. The policy that SPAR uses to choose between these options is to always attempt to satisfy a goal by adding appropriate constraints. Only if this fails will a new action be added. The reason for this is that it is always cheaper to constrain an action than to add an action, both in the planning stage *and* in the execution stage. In the planning stage, the introduction of a new action will result in the addition of its preconditions to the list of pending goals. Furthermore, the new step could possibly undo previously satisfied goals.

Ensuring the satisfaction of goals proceeds in two steps: establishing the goal and dealing with actions that can clobber the goal after it has been established. Once a goal has been selected from the list of pending goals, the planner looks for some action which has already been added to the plan that could possibly

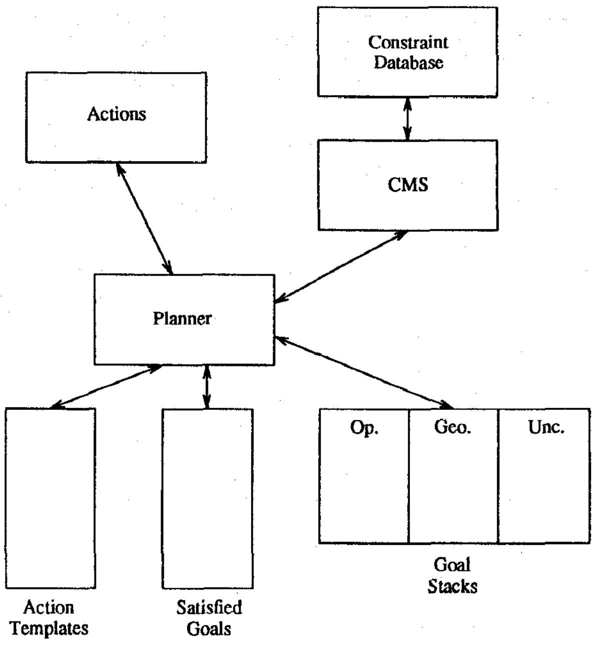


Fig. 3: Block diagram of SPAR.

establish that goal. If such an action exists, then the planner instructs the CMS to add the constraints which are necessary to ensure that the action will establish the goal. If these constraints are consistent with the current constraint database, planning proceeds. If not, then the planner selects an action to add to the plan (from its repertoire of possible actions) which can be constrained to establish the goal. The necessary constraints on that action are then added to the constraint database.

Once a goal has been established, the planner must ensure that some other step in the plan does not undo (clobber) the goal before the time at which it must be satisfied. This consists of locating potential clobberers and either constraining them to occur before the establishing action, constraining them to occur after the time at which the goal must be satisfied, or, adding an action which re-establishes the goal (such that the action occurs between the clobberer and the time at which the goal must be satisfied). The order in which these options are listed is also the order in which they are tried. Again, adding actions to the plan is the last resort.

When dealing with uncertainty goals, there is a slight difference. If there is no way for the planner to satisfy an uncertainty precondition, the planner does not backtrack. Since we represent uncertainty in the world using bounded sets (e.g. the X location of an object would be represented as $X \pm \Delta X$), it is quite possible that the actual errors in the world description will be small enough that the plan can be executed without failure. Thus, when uncertainty preconditions cannot be satisfied, rather than scrap the current task plan, our planner makes a record of the goals which were not able to be satisfied and the possible assembly errors that might occur if the errors in the world description are in fact worse than those which can be tolerated. This information can then be used to aid in error recovery during the plan execution stage.

In addition to noting the violated uncertainty preconditions, the planner also posts a verification procedure which will be used at execution time to determine whether the action succeeded or failed. This is some type of sensing operation, which is chosen based on the action and on the particular uncertainty precondition which was left unsatisfied. For example, if the action is grasping an object, and the precondition which ensures that the object is between the fingers is violated, a simple query of the manipulator can be made, to determine whether the object is in its grasp. SPAR associates one verification strategy with each uncertainty precondition *a priori*. These are generic sensing operations which define the type of verification to be performed in terms of plan variables. At plan time, these variables must be instantiated. For example, the width of an object is used to determine the opening width of the gripper which would verify that the object had been grasped successfully.

Merely recognizing that an error has occurred is not sufficient, because there may be a number of potential errors associated with any single uncertainty precondition. For this reason, the planner not only prescribes verification strategies, but it also associates possible errors with the possible verification results. Again, these correspondences are enumerated *a priori* in terms of plan variables. The planner instantiates these variables so that when the verification procedure is executed, the precise error is determined.

After the planner has established a verification procedure, it posts a recovery plan for each of the predicted errors. This consists of determining the appropriate values with which to instantiate generic recovery plans which have already been developed. These plans are generally very simple, and are intended only as local solutions to error recovery. When errors occur that have global effects (i.e. the

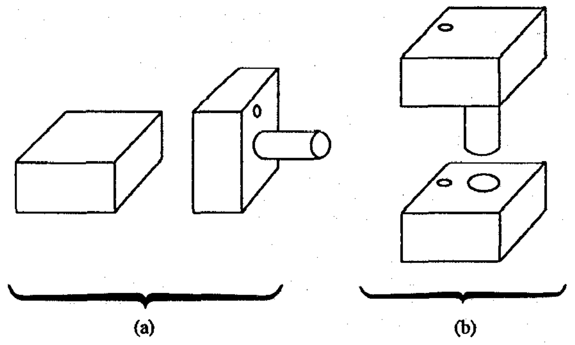


Fig. 4: a) initial state, block is face down. b) just prior to goal of inserting peg into block.

error affects more than just the action which failed), the system must "start over", first by determining the world state using the sensory system, and then developing new task plans to achieve the unfulfilled system goals.

2.1 An Task Planning Example

SPAR's flow of control is best illustrated by an example. Consider the assembly task shown in Fig. 4. The assembly goal is to have the peg inserted into the block so that the small hole in the block is aligned with the hole in the peg's base. The user specifies this goal by a statement of the form:

(assembled peg block ?M-surfaces ?TM ?Va)

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

In order to satisfy the assembly goal, SPAR examines its possible actions, and selects the *assemble* action shown in Fig. 1. Of course, the assemble action has both operational and geometric preconditions which must now be considered, so the planner pushes these onto the appropriate goal stacks. This is illustrated in Fig. 5.

```
OPERATIONAL GOAL STACK
        nil
GEOMETRIC GOAL STACK
        (C4 Step-2
                (reachable grasp-1 ?Position)
                (part-location peg ?Position))
        (C2 Step-1
                (member grasp-1 (pg1 pg2 ... pgn))
                (holding peg grasp-1))
        (C3 Step-1
                (in-position-class ?Pos2 (bp1 bp2 ... bpm))
                (part-location block ?Pos2))
UNCERTAINTY GOAL STACK
        (C5 Step-2
                (<= ?dx (- GRIPPER-WIDTH PEG-WIDTH))
                (part-location-error peg ?dx ?dy ?dz ?dr))
ACTION LIST
        (Step-2 (pickup peg grasp-1))
        (Step-1 (assemble peg block
                ((peg-face) (block-face)) trans-1 vec-1))
```

Fig. 5: Goal stacks after addition of assemble action.

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

```
(?C3 ?Step-id
        (in-position-class ?position ?Position-list)
        (part-location ?Obj2 ?position))
```

SPAR associates a set of stable positions with each object, where, by stable position, we mean an orientation in which the object will rest naturally on the table. In order to mate two objects, SPAR requires that the stationary object be in one of its stable positions which does not have the mating features of that object in contact with the work table. When the planner adds the assemble action to the plan, it instantiates the variable ?Position-list to a list of the object's stable positions which meet this condition. Note that the list of stable positions is actually a list of pointers to the data structures for the stable positions.

This same kind of instantiation takes place for the precondition

```
(?C2 ?Step-id
        (member ?Grasp ?Grasp-list)
        (holding ?Obj1 ?Grasp))
```

In our system, grasping configurations specify not only the geometric configuration which is used to grasp the object, but also the set of object features which are obscured by the grasp. When the planner adds the assemble action, it instantiates the variable ?Grasp-list to be the set of grasping configurations which do not obscure the mating features of the object.

Fig. 5 shows the instantiated versions of the preconditions, as they appear on the goal stack. Note that the variables used to identify the preconditions, ?C1,?C2, and ?C3, and the action to which the preconditions correspond, ?step-id, have also been instantiated.

The only operational goal is that the gripper be holding the peg in some valid grasp (remember that at the operational level, the planner is not concerned with the grasp beyond this condition). Since it is not possible to merely add a constraint to the plan to achieve this goal (i.e. there is no existing action in the plan whose execution can be constrained so that it results in the manipulator holding the peg), the planner inserts the action (pickup peg grasp-1) into the plan, with the constraint that the pickup action must occur prior to the mating action. The preconditions of the pickup action are then pushed onto the appropriate goal stacks, as shown in Fig. 6. Note that when the planner adds this action, it instantiates the variable ?Grasp to the label grasp-1, and that this instantiation affects all appearances of ?Grasp on the goal stacks.

```
OPERATIONAL GOAL STACK
        (C1 Step-1-step
                (holding peg ?Grasp))
GEOMETRIC GOAL STACK
        (C2 Step-1
                (member ?Grasp (pg1 pg2 ... pgn))
                (holding peg ?Grasp))
        (C3 Step-1
                (in-position-class ?Pos2 (bp1 bp2 ... bpm))
                (part-location block ?Pos2))
UNCERTAINTY GOAL STACK
        nil
ACTION LIST
        (Step-1 (assemble peg block
                ((peg-face) (block-face)) trans-1 vec-1))
```

Fig. 6: Goal stacks after addition of pickup action.

The operational goal stack is now empty, and the planner turns to its geometric goals. The top goal on the geometric goal stack, C4, is for the pickup action, and it specifies that the manipulator configuration used to pickup the peg, grasp-1, be physically realizable by the robot. In order to satisfy this goal, the planner first searches for some action in the plan which adds the fact (part-location peg ?Position). If the planner finds such an action, it attempts to constrain the execution of that action so the reachable condition will be met.

Since no such action has been added to the plan, the planner checks the initial description of the world (which is represented by a null action in our system) to see if specifies* the location of the peg. If not, the planner must invoke the sensory system. Once the sensory system has determined the peg's location, the planner attempts to add a constraint on the way in which grasp-1 is chosen, so that the configuration will be reachable. In order to do this, the CMS examines the possible grasping configurations in conjunction with the position of the peg and eliminates all configurations which can not be physically achieved by the robot. This requires the use of the inverse kinematic solution of the robot, and checking each of the joint angles to make sure they are within their specified limits. Provided that there are choices for grasp-1 which are reachable, the CMS adds two constraints to the constraint database:

```
(reachable grasp-1 peg-location)
(member grasp-1 (peg-g1 ... peg-g12))
```

These two constraints indicate that grasp-1 must be chosen so that it is reachable for peg-location, and grasp-1 must also be chosen from the set (peg-g1 ... peg-g12). Specifically listing the plausible grasps is not strictly necessary, but the CMS does this in order to save time if it must later add constraints on grasp-1. Since the C4 has now been satisfied, it is moved from the geometric goal stack to the list of satisfied goals.

The next goal on the geometric goal stack, C2, specifies that grasp-1 must not obscure any of the mating features of the peg. As we have mentioned earlier, this precondition is expressed by defining the set of grasping configurations which do not obscure the mating features. Again, the planner invokes the CMS to add an appropriate constraint to the constraint set. This time, the CMS examines the grasping configurations which were not eliminated by adding the reachable constraint, and eliminates all of those which are not in the set allowed by condition C2. Note that if adding this constraint resulted in an inconsistent constraint set (i.e. no possible grasping configurations remain), the planner would insert additional manipulations.

Up to this point, SPAR has been able to satisfy geometric goals merely by adding constraints on the way in which operations are performed. In some cases, it will not be possible to satisfy geometric goals this way, and an alternative approach must be used. This is the case for the remaining geometric precondition, C3, which constrains the possible positions of the block. Consider the situation when the block is face down in the initial world state. The planner cannot add a constraint on the block's initial position, because it is a constant value which is defined by the initial world state. Furthermore, since there is no action currently in the plan which manipulates the block, SPAR cannot constrain the execution of a plan action to achieve the goal.

The next option that SPAR investigates is the addition of the action (putdown block position-1) to the plan. The value of position-1 is constrained so that no mating features of the block are in contact with the table when the block is in this position (this constraint is guaranteed to be consistent with the constraint database, since the plan variable position-1 is introduced by this step, and therefore has no other constraints associated with it). Of course the addition of this plan step introduces new goals, and so additional planning must be done. This planning, however, is very similar to the planning which must be done to pick up the peg appropriately, and so we will not bother to discuss it here. The plan steps, and the constraint database, for these four steps of the plan are shown in Fig. 7.

```
Plan Steps
        (plan-step-4 (pickup block Grasp-2))
        (plan-step-3 (putdown block Position-2))
        (plan-step-2 (pickup peg Grasp-1))
        (plan-step-1
                (assemble peg block ((peg-face-A) (block-face-B))
                        transform-1 vector-1))
Constraint Database
        (member Grasp-2 (bg1 bg2 bg3 bg4 bg5 bg6 bg7))
        (member Grasp-1 (pg3 pg8 pg10 pg12))
        (in-position-class Position-2 (bp1 bp4 bp5 bp6))
        (reachable Grasp-2 block-position)
        (reachable Grasp-1 peg-position)
        (prior-to plan-step-4 plan-step-3)
        (prior-to plan-step-3 plan-step-2)
        (prior-to plan-step-3 plan-step-1)
        (prior-to plan-step-2 plan-step-1)
```

Fig. 7: The plan steps and constraints for the assembly plan to assemble the peg and block.

Once the operational and geometric goals have been satisfied, SPAR considers the uncertainty goals. Notice that the uncertainty precondition for grasping the peg is that the uncertainty in the X-position of the peg must be less than the difference in the width of the peg and the width of the manipulator opening. Clearly this is a greatly simplified version of the actual condition which must hold, but it will serve for this example. In its first attempt to satisfy the condition, the planner searches through the list of actions which have already been added to the plan (which includes a null action to represent the initial state) to find an action whose execution can be constrained such that it will limit the uncertainty in the peg's location. There is no such action in our plan (let us assume that the initial description of the world does not satisfy the uncertainty condition).

The next attempt is to add a sensing operation to the plan. Sensing operations are treated in the same way as manipulations. In particular, they have a set of preconditions which must be met before they are applied (e.g. to perform a range scan, the manipulator must be free) and an add/delete list which specifies the sensing actions affect on the world description. The "range-scan" sensing action is shown in Fig. 1. Obviously it is impossible to predict the results of a sensing action, so the add/delete lists merely characterize the possible reduction in uncertainty for the sensing operation. For example, the part-location-error fact in the add-list of the range-scan action indicates that range-scanning an object will reduce the uncertainty in the X,Y and Z locations of the part to an amount less than 0.5 inches, and the uncertainty in the rotation about the world Z axis to an amount less than 5.0 degrees. If such a sensing operation can be found, it is inserted into the plan. If not, the planner attempts to add some robot action to the plan to reduce the uncertainty.

If SPAR cannot sufficiently reduce the uncertainty in the peg's location, it must determine a verification strategy. In this case, the verification is to check the location of the manipulator after operation is attempted. If its height is above the table by more than an acceptable tolerance, then the manipulator is determined to have collided with the object. If the manipulator is at an acceptable height, a check is made of the opening width of the gripper fingers. If it is zero (i.e. the gripper is completely closed), then the gripper has missed the object. These conditions are illustrated in Fig. 8.

Associated with each of these verification results is a local recovery plan. If the manipulator has collided with the object, then the torque on the wrist is measured to determine which of the two fingers is currently in contact with the object. This information is used to determine the direction in which the manipulator

* Currently, SPAR invokes the sensing system only in the case when "part-location" information is lacking in the initial world state. If we did not impose this restriction, the planner would invoke the sensing system to satisfy part-location goals even if the initial part location was known and manipulation was required.

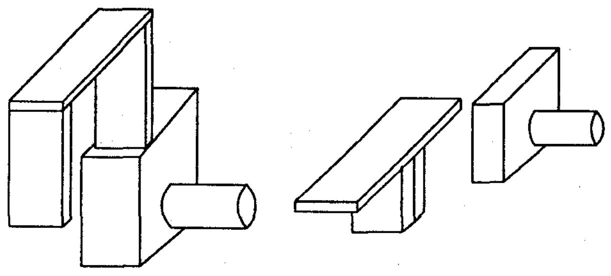


Fig. 8: Possible error conditions when the uncertainty goal is not satisfied

should move to get the object between the fingers. Finally, the manipulator is raised, moved in the appropriate direction and the grasp is retried. If there is no object in the gripper fingers, then the information about the position of the object obtained from the sensory system was very bad. In this case, a local sensing operation (often the 2D overhead camera) is applied to determine the location of the object.

3. PLANNING SENSING STRATEGIES

In today's robotic work cells, it is not uncommon to find a large number of sensors, which, in many cases, can be manipulated by the robot, so that they can be applied from arbitrary positions in the work space. This presents the system with the problem, not only of combining information from different sensors, but also with the problem of selecting pertinent sensing operations so that the system does not waste time collecting information which is not directly useful. In an earlier paper [7], we described an approach to sensor planning which was based on an aspect graph representation of objects. We then extended this work in [8] to allow the system to reason with partial evidence about the objects in the work cell.

We approach the problem of viewpoint and sensor-type selection as follows. Once the system has a working set of hypotheses (which is initially developed after application of an arbitrary sensing operation, say the 3-D range scanner), candidate sensing operations are automatically proposed and evaluated with regard to their potential effectiveness, given the current hypothesis set. This evaluation is performed as follows. For each hypothesis in the current hypothesis set, the system determines the set of features that would be observed by the candidate sensing operation if that hypothesis were correct. Using these predicted features, the system determines the hypothesis set that would be formed if these features were actually found by some sensing operation. The ambiguity of this predicted hypothesis set is calculated and noted. This is repeated for each hypothesis in the hypothesis set, and the maximum value of the ambiguities is associated with the proposed sensing operation. When a proposed sensing operation's maximum ambiguity is sufficiently low, that sensing operation is selected for application.

The object representation used in our system plays two key roles. First, it allows us to quantize the space of sensing operations. This is a result of the fact that the representation groups together sets of object features which can be viewed from a single viewpoint (such a set of features is referred to as an *aspect*). This allows us to group together all viewpoints which can observe the same aspect. Second, the representation allows us to easily determine the features of an object which will be observed by a particular sensor from a particular viewpoint relative to the object. This is done by determining which aspect of the object will be observed from the particular viewpoint, and then looking up the object features which are associated with that aspect.

Generating, and subsequently refining, hypothesis sets begins by matching sensed features to model features, and then assessing the quality of those matches. In our system, a sensed feature can be matched to any of the model features which have attributes that are similar to those of the sensed feature. The degree of similarity will determine the quality of the match. In order to reason about the hypotheses derived from these matches, the system must be able to represent its relative belief in the various feature matches. Furthermore, since an object hypothesis will correspond to a number of feature matches, the system must be able to combine the beliefs in the individual feature matches to assess its belief in an object hypothesis.

When evaluating belief in an object hypothesis, feature matches are not the only source of information. We also determine the relationships between sensed features and compare these to the relationships between the corresponding model features. As their similarity increases, so does the confidence in the corresponding hypothesis. This allows us to accumulate evidence which supports a hypothesis based on its relational consistency. It also allows us to discount hypotheses in which the relationships between sensed features are inconsistent with the corresponding model relationships, thus pruning the number of hypotheses which the system must maintain.

The final source of evidence we consider evaluates the difference between the expected and actual sensed data. Once an object hypothesis has been established, we can derive a pose transformation that expresses the position/orientation of the object if that hypothesis is correct. By using the pose transformation in conjunction with information about the sensing operation that was performed, we can determine what "should have been observed" by the sensor if that hypothesis were correct. Of course we cannot expect that sensing will always find every feature which might be present, so we assign values to each object feature which reflect the prominence of that feature. We then evaluate the quality of the object hypothesis by noting the prominence of the expected features which were (and were not) matched.

In our system, we use the Dempster-Shafer (DS) theory of evidence to implement uncertainty reasoning. One advantage of this approach is the built-in formalism for refining sets of hypotheses. We use this formalism to expand the frame of discernment as new features are found (which then lead to more specific object hypotheses). In order to assess the ambiguity in a hypothesis set, we use a measure of ambiguity which is based on the entropy measure from information theory.

4. PSEIKI: A HIERARCHICAL SPATIAL REASONING SYSTEM

We are also investigating how geometric reasoning can be used to achieve expectation-driven vision by merging image data with information contained in a graphic depiction of the expected scene. **PSEIKI**, the testbed used in this investigation, performs expectation-driven vision by integrating information contained in a graphic depiction of the expected scene with image data. The acronym PSEIKI stands for a Production System Environment for Integrating Knowledge with Images. An in-depth description of PSEIKI can be found in [1].

We are currently focusing our attention on the integration of global map information with vision data to aid navigation for an autonomous mobile robot [10]. The robot's task is to traverse a known network of sidewalks using sensor data to provide position information. For various reasons, the robot's position and orientation never is known with certainty. Therefore, for the purpose of self-location, the robot must attempt to integrate its knowledge of the sidewalk map with sensed images. PSEIKI is used to integrate the knowledge from these two sources by matching elements detected in images (*image elements*) with elements from a graphic rendition of the expected scene (*model elements*). Once the correspondence between the image elements and model elements is established, it is then possible to use camera calibration information to reduce the amount of uncertainty in the robot's position and orientation.

The match information generated by PSEIKI is expressed by labeling the image elements with the identities of the corresponding model elements; a belief value indicating the confidence of the match found is attached to each label. The Dempster-Shafer theory of evidence is used to reason about the certainty of the matches made and to overcome the problems of matching perturbed data. To overcome the exponential explosion usually associated with the Dempster-Shafer formalism, a computationally efficient variation of Dempster's rule is used to combine evidence about the labels. This variation of Dempster's rule also allows the reasoning process to exploit the hierarchical nature of the integration task. The belief value associated with the top level of the hierarchy is considered to be the confidence in the entire matching process; if this belief value does not exceed a threshold, the matches found are rejected.

As a simple illustration of PSEIKI's integration of image and expected scene information in the context of self-location of a mobile robot, consider Fig. 9.

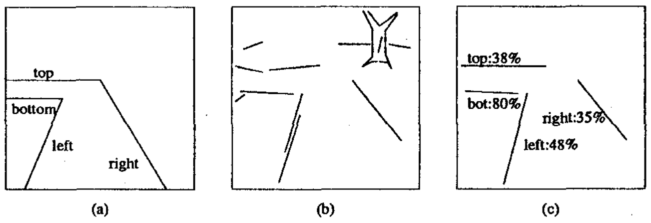


Fig. 9: This figure shows a simple example of images typical of those used by PSEIKI. The image in panel (a) shows an expected scene with edges labeled. Panel (b) shows the input to PSEIKI produced by an edge-based preprocessor. The final output of PSEIKI is shown in panel (c); the edges from panel (a) detected in the image in panel (b), along with their associated belief values are shown.

If panel (a) of this figure is a graphic rendition of an expected scene and panel (b) a depiction of the edges found in the vision data collected for the scene, then PSEIKI would produce an output similar to the one in panel (c), where the labels attached to some of the edges and their corresponding belief values are shown. For example, the label 'right:35%' means that PSEIKI has found the expected-scene edge labeled 'right' in panel (a) to be compatible with the lower right edge in panel (b) with a belief of 35%. In this case, the rest of the belief, 65%, would be apportioned either to this particular label being incorrect or to the system professing ignorance on the subject of assigning a label to this edge in the vision data. The reader might note that the edge labeled 'top:38%' actually corresponds to two edge segments in panel (b). This merger of nearly compatible edges in the vision data is one consequence of various tests PSEIKI makes for internal geometric consistencies in the vision data.

PSEIKI is not limited to integrating edge-information from the two data sources ; it is also able to match data elements at higher levels of abstraction. For example, if a region-based preprocessor is used to generate PSEIKI's input data, then PSEIKI would match the regions found in the image with the regions predicted in the expected scene. In this case, PSEIKI would also exploit edge-level information by treating the boundaries between regions as edges and matching them with edges in the expected scene. PSEIKI is also able to group low-level image elements into higher level constructs by using perceptual organization principles and by noting which elements have compatible labels. For example, if PSEIKI's low-level preprocessor provides only edge information to the system , then PSEIKI would group compatibly labeled, adjacent edges into regions. Once these higher level image elements are formed, PSEIKI can then match them with high-level model elements. The following list enumerates the levels of data abstraction present in PSEIKI and describes the data residing on each level.

Scenes	The entire scene (expected or observed) is represented on this level of the hierarchy. The scene is defined as the union of all elements on the object level. This level provides a way of labeling multiple objects that otherwise would not be possible.
Objects	Each element on this level corresponds to a distinct physical object. An object is defined by its boundary faces.
Faces	The elements on this level represent the polygonal faces that form boundary representations of the observable portions of objects. In image data, a face corresponds to a region in the image. A face is defined by the edges which form its border.
Edges	These elements represent edges detected in the sensor data; they are used to form the boundaries of the faces in the hierarchy. The endpoints of these elements are defined by vertices.
Vertices	The vertex-elements are the endpoints of the edges. The location of the vertices can be expressed either in world or image coordinates.

Initial matches between image data and model data are formed by noting geometric relationships between image elements and model elements. For example, an image-edge will be matched with the model-edge that comes the closest (in some sense) to lying along the same line. To find the match partner of an image-edge, PSEIKI measures the degree of *collinearity* between the edge and all the model-edges with which it could possibly match; it then chooses as the match partner the model-edge with which the image-edge is most collinear. The belief of the match made then is set to the degree of collinearity between the two edges.

After the initial matches are made, geometric constraints between image elements are used to update the belief in the matches found. In general, two metrics are required to measure the degree to which image-elements meet these constraints. The two metrics must provide measures of *compatibility* and *incompatibility* between image-elements. The compatibility metric provides evidence that an element's label is correct; conversely, the incompatibility metric provides evidence that an element's label is incorrect. For example, two edges that have the been matched with the same model-edge should lie approximately along the same line. Thus the compatibility metric for edge-elements with the same label, *collinearity(edge1, edge2)*, measures the degree to which the two edges lie along the same line. This collinearity metric is closely related to the measure used to establish initial edge labels, but it is not identical to that measure. The edge-level incompatibility metric, *noncollinearity(edge1, edge2)*, measures the degree to which two edges do not lie along the same line. These two metrics are used to update the belief in an edge's label in the following manner: If we assume that two edges, E_1 and E_2, both have label E_A, then the confirmatory evidence that E_2 would provide for E_1's label would be defined to be

$$\text{collinearity}(E_1, E_2) \times \text{belief}_{E_2}(E_A)$$

Likewise, the disconfirmatory evidence in E_1's label provided by edge E_2 would be

$$\text{noncollinearity}(E_1, E_2) \times \text{belief}_{E_2}(E_A)$$

where $\text{belief}_{E_2}(E_A)$ represents the amount of belief in edge E_2's label, E_A.

If two elements are matched with different model elements, a rigid motion transform is applied to one of them before the (in)compatibility metrics are applied. This transformation has the effect of enforcing relational constraints between the two data elements. For example, if image-edges E_1 and E_3 are thought to correspond to model-edges E_A and E_B, respectively, then the measure of compatibility between E_1 and E_3 would be defined as

$$\text{compatibility}(E_1, E_3) = \text{collinearity}(E_1, T_{E_B \to E_A}(E_3))$$

where $T_{E_B \to E_A}$ is the rigid motion transformation that makes model-edge E_B collinear with model edge E_A. In other words,

$$\text{collinearity}(E_A, T_{E_B \to E_A}(E_B)) = 1.0$$

Applications of transformations in this manner allows us to implement the notion that for edges E_1 and E_3 to be compatible, the same geometric relationship should exist between them that exists between model-edges E_A and E_B, if E_A is the label given to E_1 and E_b the label given to E_2.

The compatibility and incompatibility metrics must, of necessity, be different at different levels of abstractions. At the face level, for example, a metric that is used to compute the incompatibility between two faces on the data panel measures the overlap between them normalized by the average area of the two faces.

A most important aspect of evidential reasoning in PSEIKI is the propagation of beliefs up and down the abstraction hierarchy. The propagation of belief values towards the higher abstraction levels is based on the rationale that any evidence confirming a data element's label should also provide evidence that its parent's label is correct. Propagation of beliefs to lower levels is based on the intuitive idea that if, say, a face is mislabeled, then all its constituent edges are also most likely mislabeled.

PSEIKI is implemented as a blackboard expert system [12,13], in OPS83 [5]. The blackboard architecture was chosen because the success of blackboard systems in other problem domains has proven the power of the paradigm, particularly in domains that can be broken down hierarchically. In its present configuration, the system contains two panels, each with five levels. Each panel holds the data from a single data source and the abstraction hierarchies derived from that data. One panel, called the *model panel*, holds the expected scene information and the other panel, called the *image panel*, contains image data. Each level in the blackboard corresponds to one of the levels of data abstraction discussed earlier. Thus each blackboard panel contains the following levels to represent data: scenes, objects, faces, edges and vertices. At a given level, each element, except for those of type vertex, is defined by a finite collection of elements at the lower level. Fig. 10 shows PSEIKI's current architecture.

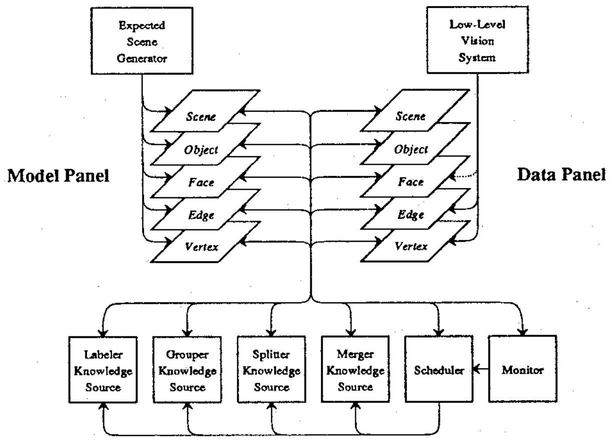


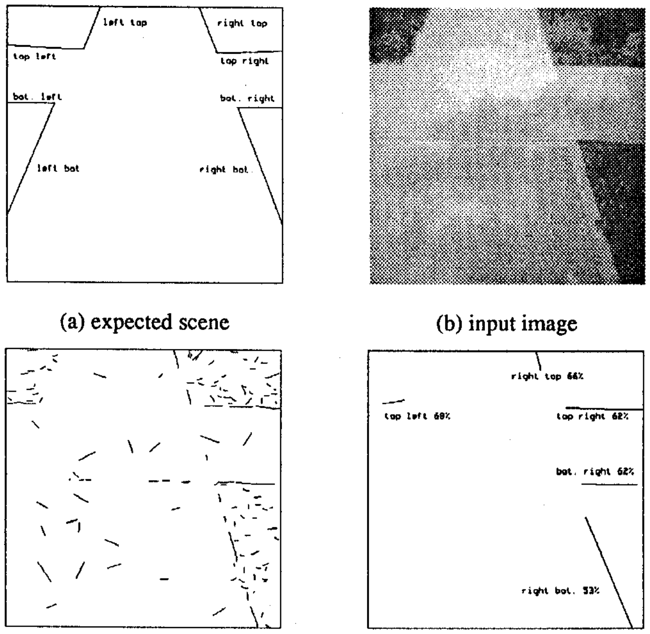
Fig. 10: The current configuration of PSEIKI's architecture.

PSEIKI has four main knowledge sources (KS's) that it uses to establish correspondences between elements from independent data sources: the *labeler*, *grouper*, *splitter*, and *merger*. The grouper KS determines which image elements in the lower levels of the hierarchy should be grouped to form an image element on a higher level. The merger KS also groups elements; however, its job is to merge multiple elements on a single level of the blackboard into an element at the same level. The splitter KS performs the opposite action of the merger KS; it splits a single element on the blackboard into multiple smaller elements. The labeler KS has the responsibility of establishing correspondences between image elements and model elements and to accumulate evidence on the validity of those labels. Each of these KSs can operate on any level of the blackboard by using level specific actions.

Input data is deposited directly onto the blackboard by the low-level systems that generate it. The type of processing performed by the low-level systems determines the levels on which data is deposited. Model data is deposited onto all levels of the blackboard because perfect knowledge of the expected scene is always available. If an edge-based preprocessor is used to produce data for PSEIKI, then the data generated by the vision system is deposited onto the vertex and edge levels of the data panel. However if a region-based system is used, then the input data also is deposited onto the face level of the blackboard. (In Fig. 10, the dashed line going from the low-level vision system to the face level of the data panel indicates that only some types of systems deposit data on that level.)

Figure 11 shows the results of PSEIKI's processing when applied to an image typical of what would be seen by a sidewalk-navigating mobile robot with downward-slanted cameras. Panel (a) of this figure shows the edges representing the expected scene and panel (b) shows the actual image being processed. Note that the expected scene and the observed image are significantly misregistered. Two of the major edges in the expected scene, in the lower left, are missing entirely in the observed image. The reader should also note the presence of shadow edges in panel (b). The symbolic input to PSEIKI, produced by an edge-based preprocessor, is shown in panel (c).

The final result produced by PSEIKI consists of labels with associated belief values attached to entities at the edge level and higher levels on the image panel on the blackboard. For example, in panels (a) - (c), if the element at the scene level (the highest blackboard level) with maximum belief is selected and its component edges are displayed, panel (d) results. This figure shows the edges, their labels, and associated belief values for the scene interpretation that PSEIKI found



(a) expected scene (b) input image

(c) output of the preprocessor (d) output of PSEIKI

Fig. 11: This figure shows an example of PSEIKI's processing in a mobile robotic context.The expected scene is shown in panel (a); it shows the edges expected in the image and their labels. Panel (b) shows the image input to the low-level preprocessor; panel (c) shows the preprocessor's output. PSEIKI's output is shown in panel (d); the detected edges, their labels and associated belief values are displayed there.

most believable. The percentage value associated with a label indicates PSEIKI's belief in the correctness of the label. For example, PSEIKI has a belief of 0.53 that the lower right edge can be matched with the right-bottom edge of the expected scene. This amount of belief indicates that, at a belief level of 0.47, PSEIKI believes that the edges were mismatched or that the system is ignorant about the validity of the match made.

5. OTHER RELEVANT RVL CONTRIBUTIONS

For a complete system, task planning must be coupled to what might be called *sensor guided execution units.* For example, if at execution time the task planner calls for one part to be inserted into another part, the task planner should then temporarily hand over control to a parts mating process with its own built-in error detection and recovery procedures. In this manner, a clean separation can be achieved between the more global task-level considerations embedded in the task planner, and the more local fine-motion strategies required for successful assembly, especially when the assembly must be carried out under tight tolerance conditions.

In [6], we demonstrated an implementation of force/torque control in which the assembly steps are decomposed into various phases, each phase being characterized by a unique error detection and recovery strategy; also, each phase consists of either a guarded move, in which the robot is servoed with respect to forces along the direction of motion of the end-effector, or a compliant move, in which the servoing is with respect to forces perpendicular to the direction of motion. Automatic error detection and recovery is particularly important to force/torque guided manipulation because force/torque sensors are highly susceptible to mechanical vibrations, inertial effects, and electrically induced distortions. To give the reader an idea of what we mean by error detection and recovery, we will consider the compliant move that is supposed to "drag" the peg over the surface containing the hole until the bottom face of the peg is directly over the hole; one of the termination conditions for this move is the zeroing of the force component along the axis of the hole. During the compliant move, it will be all too easy to receive a false indication of this termination condition due to some noise or artifactual condition associated with the force/torque sensor. Therefore, every sensed termination condition must be checked for the possibility of error, and, if an error is detected, a recovery plan instituted. In this case, the error can be detected by the unsuccessful execution of a small guarded move to a point slightly below the height of the surface containing the hole. If an error is detected, the recovery could consist of recalculation, of the basis of measured torques, of the direction of travel of the peg with respect to the hole.

In [11], we discussed an integrated system for coupling CAD with robotic manipulation. We pointed out that for this purpose we needed a representation of objects that would be intermediate between CAD representations on the one hand and those used for vision processing on the other; we called these intermediate representations *sensor-tuned representations.* A sensor-tuned representation is a data structure designed for the specific purpose of facilitating the recognition of objects and the determination of their positions and orientations from data provided by a given sensor or sensors. Therefore, by definition, a sensor-tuned representation for structured-light 3-D vision sensor is different from that for a 2-D photometric sensor or a tactile sensor. The sensor-tuned representations shown in [11] were for structured-light 3-D vision data and for object recognition schemes implemented in Prolog.

It is also possible to use vision as input for a sensor-guided-motion-execution-unit. In [2,3], we showed how structural stereopsis could be used for the depth perception and manipulation of rod like objects. In structural stereopsis, a scene is characterized by a set of primitives and their interrelationships, and binocular fusion for the purpose of depth perception is carried out by matching primitives under the constraints generated by their interrelationships. In our work, we used parametric structural descriptions of scenes. In such descriptions a parameter is used to measure the strength with which a set of primitives are participating in a relationship. In our more recent work [16], we have carried out a rule based implementation of binocular stereopsis. The rule based system represents an integration of the Marr-Poggio-Grimson technique with the matching of higher level percepts. It is now realized that the Marr-Poggio-Grimson method all by itself is incapable of generating range information that would be dense enough to be useful for robotic manipulation.

REFERENCES

[1] K. M. Andress and A. C. Kak, "Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System," *The AI Magazine* Vol. 9, No. 2, pp. 75-94, 1988. [A more up to date exposition can be found in: K. M. Andress and A. C. Kak, "The PSEIKI report - Version 2," School of Electrical Engineering, Purdue University, Technical Report TR-EE 88-9, 1988].

[2] K. L. Boyer and A. C. Kak, "Structural Stereopsis for 3-D Vision," *IEEE Trans. on PAMI* Vol. PAMI-10, pp. 144-166, March 1988.

[3] K. L. Boyer, A. J. Vayda and A. C. Kak, "Robotic Manipulation Experiments Using Structural Stereopsis for 3-D Vision," *IEEE Expert*, pp. 73-94, August 1986.

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

[5] C. L. Forgy, "The OPS83 User's Manual, System Version 2.2," Production Systems Technologies, Inc. 1986.

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

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

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

[9] S. A. Hutchinson and A. C. Kak, "A Task Planner for Simultaneous Fulfillment of Operational, Geometric and Uncertainty Goals," School of Electrical Engineering, Purdue University, Technical Report TR-EE 88-46, September 1988.

[10] A. C. Kak, B. A. Roberts, K. M. Andress and R. L. Cromwell, "Experiments in the Integration of World Knowledge with Sensory Information for Mobile Robots," *Proc. of the IEEE Int. Conf. Robotics and Automation*, Vol 2., pp. 734-741, 1987.

[11] A. C. Kak, A. J. Vayda, R. L. Cromwell, W. Y. Kim and C. H. Chen, "Knowledge-Based Robotics," *Int'l Journal of Robotics Research*, Vol. 26, No. 5, pp. 707-734, 1988.

[12] P. H. Nii, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of the Blackboard Architectures," *The AI Magazine*, pp. 38-53, Summer 1986.

[13] P. H. Nii, "Blackboard Systems from a Knowledge Engineering Perspective," *The AI Magazine*, pp. 82-106, Fall 1986.

[14] J. Pearl, "Fusion, Propagation, and Structuring in Bayesian Networks," *Artificial Intelligence*, Vol. 29, pp. 241-288, 1986.

[15] W. Swartout, ed., "Workshop Report: DARPA Santa Cruz Workshop on Planning," *The AI Magazine*, pp. 115-130, Summer 1988.

[16] S. Tanaka and A. C. Kak, "A Rule-Based Approach to Binocular Stereopsis," School of Electrical Engineering, Purdue University, Technical Report TR-EE 88-33, July 1988.