Integrating Sensing, Task Planning, and Execution for Robotic Assembly

Chao-Ping Tung, *Member, IEEE,* and Avinash C. Kak, *Member, IEEE*

Abstract— **This paper deals with enhancing the level of autonomy in a robotic work cell. With that mission in mind, we present here an integrated framework for the sensing, the planning, and the execution aspects of assembly. In experimental demonstrations of this system on a PUMA762, we can now throw objects randomly into the workspace of the robot and the robot then automatically synthesizes a manipulation plan that includes the operations of sensing, grasping, and regrasping. Each operation is invoked only when it is deemed necessary for the successful execution of assembly.**

I. Introduction

GIVEN the rather frequently expressed doubt these days about the merit of injecting artificial intelligence into robotics, we need to demonstrate at the very outset what our system can really do by way of useful experiments. The reader will probably agree with us that if the industrial revolution that was supposed to be heralded by robotics is ever to come to fruition, robotic assembly will have to be shown to be as free as possible of the accouterments of hard automation. We believe that experiments demonstrating robots performing complex assemblies from parts placed in random positions and orientations would serve as automation benchmarks for the future. Evidently, such robots would have to synthesize their own manipulation sequences, invoking sensors only when necessary, using fixtures only when needed, etc.

Consider the following experiment that we can now do effortlessly on our robot, a PUMA 762 arm. We throw the parts of an assembly in front of the robot so that they land on the work surface in random poses and random locations within an area that is kinematically accessible to the robot. Admittedly, at this time, these parts are simple (Fig. 1), but still we believe that for a robot to come up with all the grasping, putdown, pickup, regrasping, mating, fixturing, and sensor-selection operations automatically, and without human intervention, is no small accomplishment. Furthermore, the system is capable of learning assembly goals automatically—the human user can specify assembly goals to the robot by wearing a DataGlove and performing the task. The system observes the assembly motion through the signals received from the glove and then deduces the assembly goal. Fig. 2(b) is a snapshot of the system learning the assembly goal of Fig. 2(a). Shown in Fig. 3 is a sequence of photos showing a human operator throwing two parts in front of the robot and the robot then assembling the parts to form the assembly shown in Fig. 2(a) using an automatically generated manipulation program. Further details describing the sequence of steps shown in Fig. 3 are presented in the Appendix.

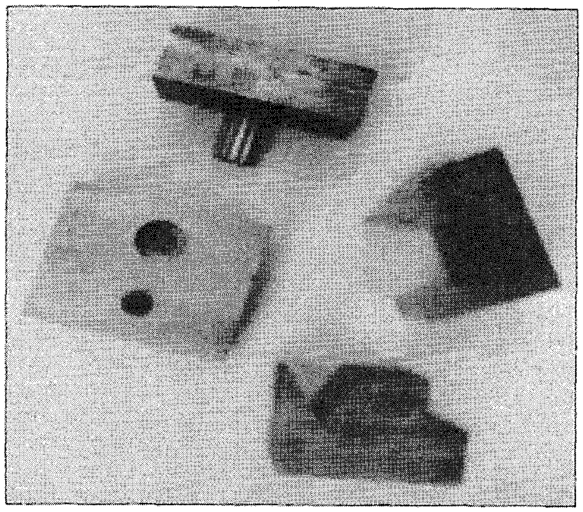


Fig. 1. Parts for assembly.

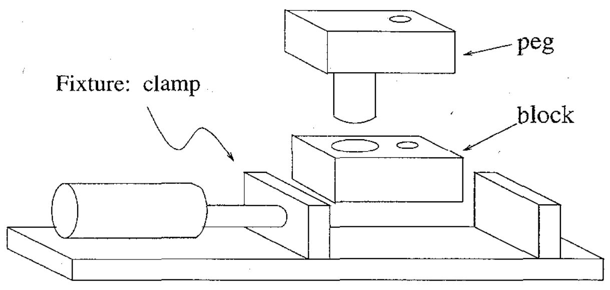
This accomplishment is an advance—conceptually, theoretically, and experimentally—over SPAR, an earlier system to come out of our laboratory [18]. For one, the current system, called the integrated robotic assembly system (IRAS), incorporates fixturing in its planning process; that means the new planner has to reason about how and when to place objects in the fixture. Also, in SPAR a human had to supply to the planner the initial positions and orientations of all the parts needed for the assembly, which necessitated that these objects not be disturbed during the execution. In the new planner reported here, the planner automatically adds sensory requests to search the work area for the needed parts and takes into account the fact that prior actions may disturb the poses of other objects. Another shortcoming of SPAR was that it did not take space into account specifically during the planning process. Although SPAR did check the kinematic feasibility of reaching a part for pickup, it did not account for the kinematic constraints for the approach point, nor did it take into account explicitly the locations for object placement for putdown operations—an important consideration for regrasping. Thus, the regrasping steps generated by SPAR were not always

Manuscript received November 1, 1994; revised August 23, 1995.

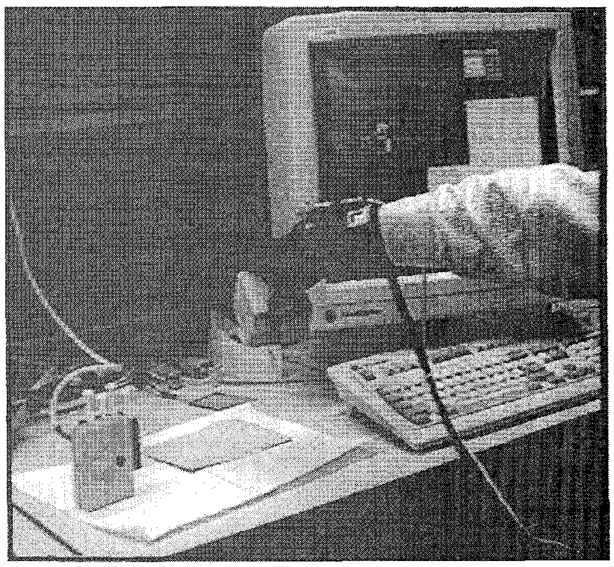
C.-P. Tung was with the Robot Vision Laboratory, Purdue University, West Lafayette, IN 47907-1285 USA. He is now with NEC America, Irving, TX 75038 USA.

A. C. Kak is with the Robot Vision Laboratory, Purdue University, West Lafayette, IN 47907-1285 USA.

Publisher Item Identifier S 1042-296X(96)02546-3.



(a)



(b)

Fig. 2. A snapshot of the system learning an assembly goal. (a) An assembly example involving a fixture. (b) User specifying the assembly of (a) using a DataGlove.

feasible for the robot. Even if regrasping is not necessary, the grasping step may still fail because the robot cannot reach the approach point for pickup. We have rectified this deficiency in IRAS. There exist many additional important differences between SPAR and IRAS, many of them dealing with the issue of computational efficiency; some of these are included in the discussion that follows.

In the rest of this paper, in Section II we give the reader an overview of the reasoning and control architecture of IRAS that makes possible the kinds of experiments we discussed above. Section III then surveys the related work published to date. Since representation is a key to how efficiently planning and reasoning can be carried out, Section IV discusses how the various entities of IRAS are represented. Section V goes into the issue of how the destination pose of an object that is supposed to be fixtured can be computed. In Section VI, we then take up the subject of uncertainties associated with the various entities. Section VII presents the assembly task planner; the purpose of this planner is to generate that part of the plan that is independent of highly variable parameters such as the initial positions and orientations of the objects to be assembled. Subsequently, Section VIII shows how the partial plan of Section VII can be augmented and further constrained to yield a truly executable plan. In Section IX, we then proceed to sensory and error-recovery issues. Finally, in Section X we present the conclusions.

II. Overview of IRAS

Shown in Fig. 4 is a top-level view of the architecture of IRAS. The system is implemented as communicating processes in PROLOG, C, and LISP, all running in a UNIX environment. The planning system, running on a Sun workstation, controls a PUMA 762 robot. A force/torque sensor and a pneumatic fixture are used to aid the assembly. Visual sensing takes place with the help of two cameras; one, mounted permanently overhead, has a view of the entire work area in front of the robot and the other, mountable in the gripper, is used for a closer look at the objects on as-needed basis.

The system interacts with the user via the *System Interface*, which is a graphics interface that allows the user to enter new assembly *Goals* and to update the *Database* of the system. The database represents the system's knowledge of the work cell. Some of this knowledge is static in time, such as about the tools that the system can use, the locations and capacities of various storage bins, etc. This information constitutes part of the knowledge that is initially provided to the system. Other kinds of information, such as the positions and orientations of the objects in the work cell, are dynamic. These data are obtained by the system as it acts to achieve its goals. In other words, the contents of the database do not necessarily represent a complete knowledge about the world, but only those parts of the world that are deemed important by the system.

The system satisfies the assembly goals by using an *assembly task planner*. It is a nonlinear, constraint posting planner [8] that uses the geometric specifications of the final assembly and the available tools of the work cell to generate flexible plans consisting of sensory-motor actions. These plans are general, in the sense that they list the possible poses and grasps that would permit a given assembly to be completed successfully. They are also flexible because the plans are generated independently of the initial positions and orientations of the parts. Thus, these plans are also reusable, i.e., creating multiple instances of the same assembly requires the planner to only generate one plan, assuming that the available tools and fixtures have not changed.

The plans generated by the assembly task planner are particularized to the specific conditions in the work cell by the *assembly execution planner*, which also schedules the appropriate action for execution. An action could either be a sensory request or an assembly operation. The sensory requests are issued to the *sensors* to obtain information about the current state of the work cell, such as the positions and orientations of certain parts. Whenever the physical limitations of the robot or the geometric constraints imposed by the initial position and orientation of an object prevent a planned action from being carried out immediately, the assembly execution planner augments the plan by generating the regrasping steps necessary to take the object from its current pose to that needed for assembly.

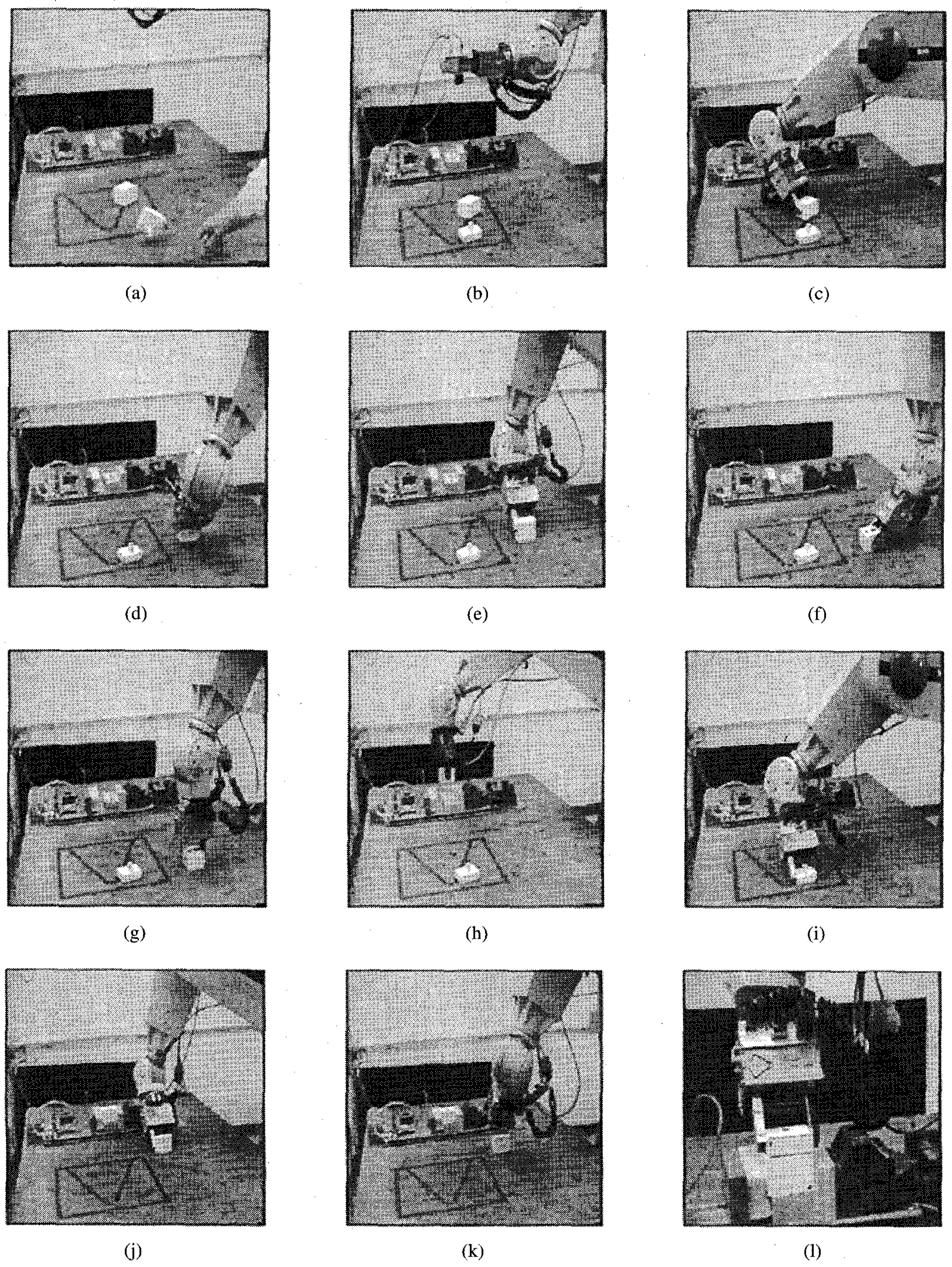


Fig. 3. Shown in (a) is a human throwing the parts in front of the robot such that their poses and locations are random. The sequence of photos from (b) to (l) shows the robot carrying out the assembly using an automatically generated manipulation program. Further details on the various frames shown here can be found in the Appendix.

The actions that are scheduled for execution are carried out by the *execution-monitor*. The execution-monitor accomplishes this by translating the actions into their corresponding actuator commands, sending them to the *effectors* for execution, and monitoring the execution process with the sensors. It is also used in conjunction with the assembly execution planner to perform error recovery.

III. Related Work

Traditional approaches to planning for assembly have often been guided by the STRIPS assumption that the only agent that can bring about a change in the world is the robot, and that only the predicates specified in the descriptions of actions can modify the world state [12], [1]. The inputs to these kinds of systems usually consist of the desired goals and a complete description of the initial state, including the initial poses of the objects. Typically, these planners produce complete plans off-line, with the final plan being either a fixed sequence of actions, a partially ordered sequence, or a hypergraph containing all possible assembly sequences [12], [18], [28], [10]. The sequence of preprogrammed plan steps thus generated is then executed by some sort of an execution-

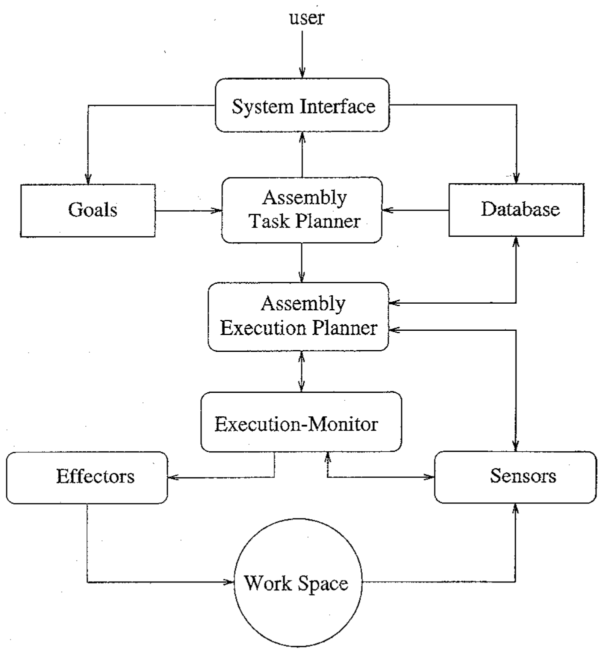


Fig. 4. Block diagram of IRAS.

monitoring module. However, on account of the dynamic nature of a work cell and the uncertaintities associated with the locations of objects, this approach tends to be inefficient and ineffective.

Regarding the matter of scheduling actions needed for an assembly, some researchers have proposed adaptive approaches that supposedly increase the flexibility and robustness of an assembly system [14], [33]. In these architectures, a complete plan, generated off line, is downloaded into an adaptive scheduler unit for execution. Developers of these architectures have shown by simulations that the plans so devised can cope with unexpected events, like delays in the arrival of parts, by selecting applicable alternative actions. Being only action schedulers, these approaches do not address important issues such as planning for sensory requests, use of fixtures, and regrasping. Furthermore, because these systems have separated planning from execution, their ability to deal with unexpected events is limited, at least in the sense reported in [27] and [31].

Another approach to enhancing a robot's ability to deal with unexpected events in the environment is to use reactive execution. This approach allows for fast (and supposedly robust) action in the absence of an explicit plan. Such systems first came into existence in mobile robotics [6], [7] and their likes are now beginning to be reported for arm robots [22], [23]. In general, these system act as dynamic schedulers of actions and do not address the issue of automatic generation of the initial task plan.

IV. Representational Issues

It has long been recognized that the power of any symbolic reasoning and planning system depends ultimately on the representations employed for the various abstract and concrete entities involved. Yet, there do not exist any formal procedures for selecting the "best" representations. All one can do is to learn from the representations used by previous researchers and conjure up new ones to advance the state of the art.

IRAS needs representations for actions, sensory-requests, goals, plans, nodes that inhabit the search space of plans, etc. For actions we have used, with minor modifications, the representation in SPAR [18]. Shown in Fig. 5 is the **fixture** action. The first part of the action consists of a rule that tells the system how to instantiate the various planning variables and that invokes a set of initial constraints. The second part of the action is the action template, in the style of a STRIPS action, that consists of a set of preconditions, an add list, and a delete list. The first line of the rule says that this action can be used to satisfy a goal of type **fixtured(Object, Fixture, ContactFacePairs, Pos, FPos)**, where the first parameter of the goal, **Object**, is instantiated to an identifier for the object that needs to be fixtured; the second parameter, **Fixture**, is set to the identity of a fixture in the work area; the third parameter, **ContactFacePairs**, is set to the pairs of object and fixture faces that are in contact after the object is fixtured; the parameter **Pos** is set to the pose of the object as it is set down in the fixture;

Rule:

to satisfy the goal:
fixtured(Object, Fixture, ContactFacePairs, Pos, FPos)

generate unique identifiers for:
ActionId, G1, G2

compute the placement and fixtured pose of the object:
PTm, FTm, Wo

compute the valid grasps for the fixture operation:
OkGrasps

retrieve the fixture's approach point:
ApproachPt

add to the constraint network the geometric constraints:
same_pose(Pos, PTm), same_pose(FPos, FTm)

Template:

action id:
ActionId

action name:
fixture(Object, Fixture, ContactFacePairs, Pos, FPos)

preconditions:
```
goal( G1, ActionId, [ available( Fixture, yes ) ], [], [] )
goal( G2, ActionId, [ holding( Object, Grasp, Location ) ],
      [ member( Grasp, OkGrasps ) ],
      [ reachable( Grasp, Pos, ApproachPt ) ])
uncertainty-check( fixture, [ Fixture, Grasp, PTm, Wo ] )
```

add list:
fixtured(Object, Fixture, ContactFacePairs, Pos, FPos)
gripper(open)
available(Fixture, no)

delete list:
holding(Object, Grasp, Location)
gripper(closed)
available(Fixture, yes)

Fig. 5. Rule and template for the fixture action.

and, finally, the parameter **FPos** is set to the value of the final pose of the object after the fixture clamps are activated.

The *precondition* part of the action template uses a special syntax that was first used in SPAR. The precondition shown in Fig. 5 says that the goals **available(Fixture, yes)** and **holding(Object, Grasp, Location)** must be satisfied before the goal **fixtured** can be satisfied. Furthermore, the success of the fixturing operation depends on the uncertainty in the description of the object.[1] The arguments to the **uncertainty-check** predicate supply the necessary information for estimating the uncertainty associated with the position and orientation of the object. This predicate can be invoked during execution time to ensure that the object's location falls within the tolerance that will allow the fixture operation to succeed. The issue of uncertainty reasoning will be discussed in greater detail in Section VI.

The satisfaction of the second precondition goal of the fixture action is subject to constraints **member(Grasp, Okgrasps)** and **reachable(Grasp, Pos, ApproachPt)**, where **ApproachPt** is the approach point. The membership constraint is a *geometric* constraint, so termed because it is based entirely upon the geometric relationship between the object and the fixture. The other geometric constraints used are **in_pose_class** and **same_pose**; these are used to constrain the pose configuration of the object. As a part of satisfying the precondition goals, the geometric constraints are sent to a constraint satisfaction network with all possible instantiations for the variables involved. As explained in [8], finding mutually consistent instantiations for the variables is one of the basic tenets on which the constraint posting approach to planning is based.

The **reachable** constraint, on the other hand, depends both on the initial position and orientation of the object as well as on the physical limitations of the robot manipulator. Thus, it is a *kinematic* constraint. The reachable constraint checks to ensure that the robot is able to reach both the approach point and the desired grasp point with the planned grasping configuration. The other kinematic constraint used by the system is **mate_reachable**, which accounts for the geometric and kinematic aspects of a mating operation. The kinematic constraints are useful for specifying the set of feasible grasp configurations for a particular situation in the work cell. These constraints are not applied until execution time, as discussed further in Section VIII. A pictorial illustration of these constraints is given in Fig. 6. The rest of the action in Fig. 5 is self-explanatory.

Other actions in the system—these would be actions for **pickup**, **putdown**, **locate**, **assemble**, and **unfixture**—are similar in form to the one shown here for **fixture**.

We will now describe the nodes in the search space of plans; this should give the reader a clue as to how the search for a correct plan takes place. Each node is a partial plan containing uninstantiated variables. A node consists of five separate entities: the list of pending goals, called the GoalStack; the list of actions planned so far, called the PlannedActions; the constraint networks generated so far from the root to the node in question; the list of satisfied goals by the PlannedActions; and the rationale that shows which action was used to satisfy each goal as it was transferred from the goal stack to the list of satisfied goals.

We mentioned above that a part of the plan description at each node in the search space consists of constraint networks. Actually, there are only two networks that the planner has to keep track of. One of the networks deals with constraints pertaining to the permissible instantiations for plan variables such as grasps, object poses, object locations, etc. The other network partially orders the actions invoked so far in the search space with a *prior_to* relation. These descriptions for the two networks should give meaning to the statement that each node, being only a partial description of a plan, represents a family of plans. Therefore, any plan that does not violate the constraints in either of the networks, would be consistent with a given node.

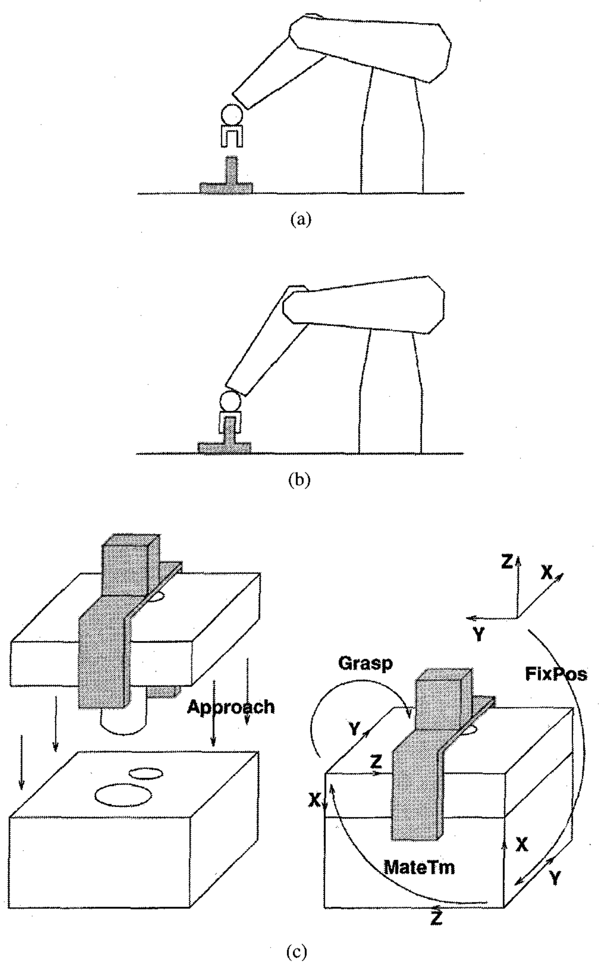


Fig. 6. Pictorial illustration of the REACHABLE and MATE_REACHABLE constraints. (a) A feasible approach point. (b) A feasible grasp. (c) A MATE_REACHABLE(Grasp, FixPos, MateTm, Approach) constraint.

[1] It has been our experience that it is not necessary to reason about uncertainties for binary assemblies for the kinds of simple objects shown in Fig. 1. However, if the uncertainties associated with the sensory reports regarding the positions and orientations of objects are large, planning of the kind reported in this paper would not succeed without an explicit accounting of the uncertainties in the manner discussed in Section VI. Due to the high quality of locational information gleaned from the vision sensors, the specific implementation of IRAS reported in this paper ignores uncertainties.

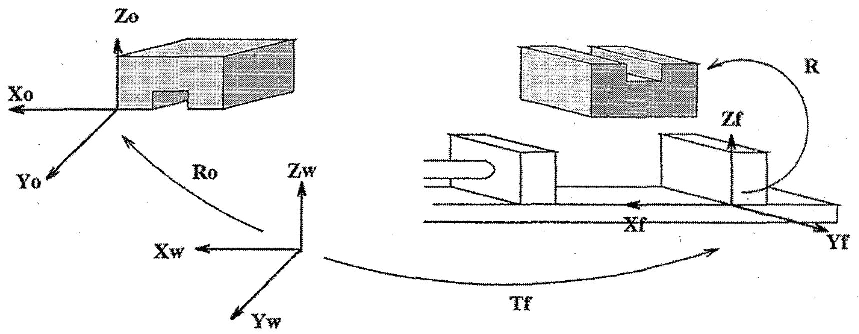


Fig. 7. The object, world, and fixture coordinates and the pose for the placement of the object in the fixture.

The successor nodes of each node, if they exist, are created by choosing the topmost goal in the goal stack and then determining whether an existing action in the PlannedActions list can be constrained to satisfy this goal. If that does not work, a new action is posted on PlannedActions and its preconditions pushed into the GoalStack. Because the addition of a new action may undo a previously satisfied goal, the planner checks the satisfied goals list, and any goal that is undone by the new action is moved to the GoalStack so that it can be resatisfied. Subsequently, the system must make sure that none of the previously posted actions will clobber the goal just transferred to the list of satisfied goals. If the goal will be clobbered, the system will invoke a declobbering strategy consisting of adding further constraints to the constraint database. For all of the different aspects of goal satisfaction and declobbering, the reader is referred to [8] for further details.

V. Object Pose Determination for the Fixture Action

Before any fixture can be used to fix the position of an object, the object must be placed within the fixture. Thus, we need to compute the object's destination pose in the fixture. In IRAS, the configuration of this pose is described by symbolic spatial relationships between those faces of the object and the fixture that come into contact with one another. Given the fixture-object contact face pairs, the planner computes a rotation matrix that is capable of transforming the object's standard pose into the destination pose in the fixture. Two noncoplanar contact face pairs are sufficient to determine the orientation of the object relative to the fixture. We employ a two-step method to determine the orientation matrix, R. First, the rotation matrix, R_1, that aligns the first of the contact face pairs is computed. Then, a rotation about the normal to this pair of planar faces, the matrix represented by R_2, is computed that satisfies contact face requirement for the other pair. This section details how these computations are carried out.

The problem of determining the rotation matrix that describes the orientation of an object in a fixture can be formulated as follows.

> Compute the rotation matrix R, which is the orientation of the object relative to the fixture, given the following information: the name of the fixture, the identity of the object, and noncoplanar contact face pairs f_1^f, f_1^o and f_2^f, f_2^o, where f_i^f and f_j^o denote fixture face i and object face j, respectively.

This information enables us to retrieve all the data needed to compute R. The following information is obtained from the fixture model database: η_1 and η_2, the unit normals of the faces f_1^f and f_2^f, respectively; T_f, the orientation and location of the fixture relative to the work cell. The object model database provides the values for the normals of the faces f_1^o and f_2^o. The object normals are then transformed to the fixture coordinate frame by premultiplying the normals by $T_f^{-1}R_0$, where R_0 is the transform that relates the object's model frame relative to the world coordinate frame. An illustration of these relationships is shown in Fig. 7.

Let ξ_1 and ξ_2 be the unit normals of f_1^o and f_2^o expressed in the fixture coordinate, respectively. Then R is simply R_2R_1, where R_1 and R_2 are as described earlier. Thus, once we have determined the matrices R_1 and R_2, we have solved the above problem. Since f_1^f and f_1^o form a contact face pair, they must have opposing normals after the object is placed in contact with the fixture. This means that R_1 can be determined by solving (1). In the discussion that follows, the coordinate frame of reference is the local fixture coordinate frame:

$$-\eta_1 = R_1\xi_1. \tag{1}$$

R_1 can be further decomposed and expressed as a product of two basic transformations.

1) Rotate ξ_1 about either the x- or the z-axis so that its y component is the same as $-\eta_1$.
2) Rotate this result about the y-axis so that another of the remaining components of the two normal vectors match.

Denoting the result of steps 1 and 2 by $\hat{R}_1$ and $\mathrm{rot}(y,\beta)$, respectively. Then $R_1 = \mathrm{rot}(y,\beta)\hat{R}_1$. Both of these steps require solving for the root of an equation of the form

$$\mathcal{F}(\theta) = a\cos\theta + b\sin\theta + c. \tag{2}$$

For example, if we want to rotate about the x-axis so that the y components of the two normals ξ and $-\eta$ match, then (3) must be satisfied:

$$\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -\sin\theta\\ 0 & \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}\xi_x\\ \xi_y\\ \xi_z\end{bmatrix} = \begin{bmatrix}-\eta_x\\ -\eta_y\\ -\eta_z\end{bmatrix}. \tag{3}$$

In this case, the values of a, b, and c of (2) are ξ_y, $-\xi_z$, and η_y, respectively. These instances of (2) will always have a solution, $\theta \in [0, 2\pi]$.

We modified the Brent algorithm in [26] to solve for the root of this nonlinear equation. This iterative algorithm employs the Van Wijngaarden–Dekker–Brent method, which allows one to find bracketed roots of a general one-dimensional function without having to compute the function's derivative. Since (2) can always be evaluated within the initial interval containing the root, this method is guaranteed to converge [3].

The rotation matrix R_2 represents the rotation of the object about the normal η_1 so that ξ_2' and η_2 would become opposing normals, where ξ_2' is obtained by applying the transformation R_1 to ξ_2. Thus, R_2 is constrained to satisfy

$$-\eta_2 = R_2 \xi_2' \tag{4}$$

where R_2 is $\mathrm{rot}(\eta_1, \alpha)$. We can visualize this process as transforming ξ_2' to coincide with $-\eta_2$. This is accomplished in four steps.

1) Transform η_1 so that it becomes parallel to the z-axis.
2) Apply this transform to ξ_2' and η_2.
3) Rotate the transformed ξ_2' about the z-axis so that it would match the transformed $-\eta_2$ axis.
4) Transform the coordinates back to the local fixture coordinate.

The transformation, $\hat{R}_2$, in step 1 is computed using the Brent method. Steps 2 and 3 are matrix multiplications and inverse tangent computations, respectively. R_2 is the result of step 4, which can be represented by

$$R_2 = \hat{R}_2^{-1} \mathrm{rot}(z, \psi) \hat{R}_2 \tag{5}$$

where $\mathrm{rot}(z, \psi)$ is the result of step 3. An example that illustrates these computations is shown in Fig. 8.

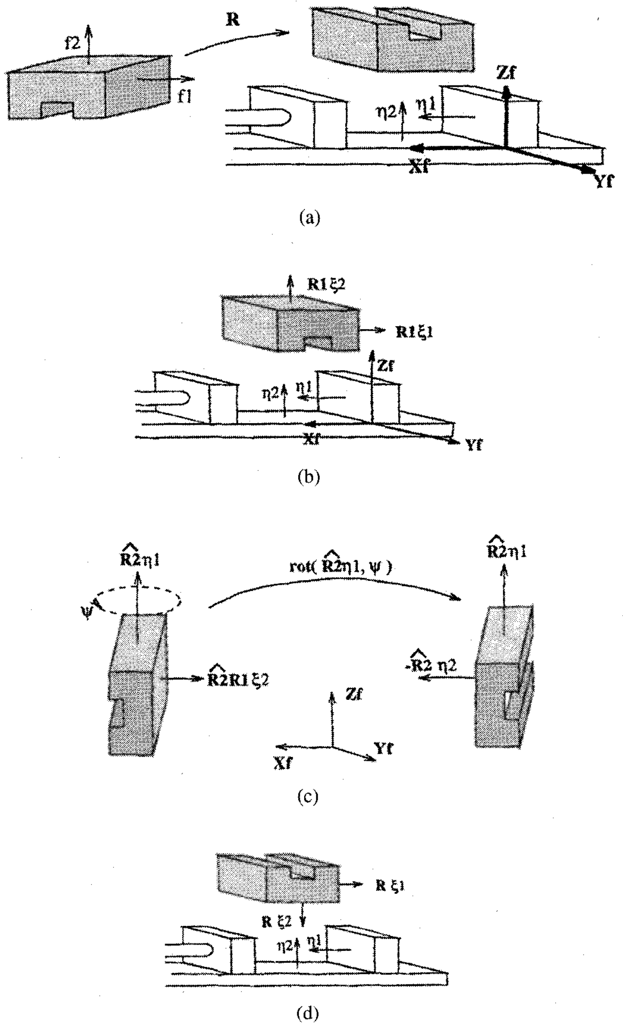


Fig. 8. Solving for R for fixturing an object with a clamp. (a) The contact face pairs for the placement of an object in a clamp. (b) Intermediate result after solving for R_1. (c) Solving for R_2. (d) Final pose of the object.

VI. Planning for Uncertainty in IRAS

The representation scheme used in IRAS is capable of accommodating the uncertainties in a robotic work cell, although, as was mentioned before, for high-quality sensory information and for simple binary assemblies, the uncertainty framework can be bypassed, as was done for the specific implementation reported here. Uncertainty is handled by using variables that take on values from bounded sets. This approach is similar to the one presented in [18] and will be reported only briefly here. The following quantities are considered to be uncertain: the position and orientation of objects, fixtures, and the gripper of the robot. In this section, we will describe the representation scheme for these uncertain quantities.

Regarding positional uncertainties, we assume that objects resting on the work table are in stable poses. Therefore, the only uncertainty in the orientation of an object resting on the table is the rotation about an axis perpendicular to the table. This axis is parallel to the Z-axis of the world coordinate in our implementation. The other uncertain quantities in the object description are the location of the object. These possible uncertainties are illustrated in Fig. 9. Homogeneous transformation matrices that are used to express the position

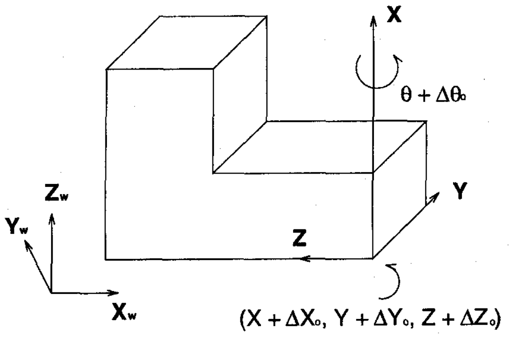


Fig. 9. Possible uncertainties in the position and orientation of an object in a stable pose.

and orientation of an object can also be used for expressing the uncertainties associated with the position and orientation. For that purpose, the entries of these matrices are expressed in terms of the uncertain variables described above. The possible

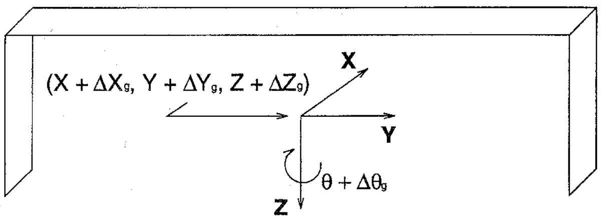


Fig. 10. Possible uncertainties in the configuration of the gripper.

positions and orientations of an object are obtained by substituting valid values for its associated uncertain variables and then combining this information with the ideal position and orientation of the object. Using this formalism, the possible positions and orientations of an object resting on a work table are given by the following equation [18]

$$\mathcal{T}_w^{o+\Delta} = \mathbf{T}_{\Delta o}\mathbf{T}_o\mathbf{R}_{\Delta o}\mathbf{R}_o \tag{6}$$

where $\mathbf{T}_{\Delta o}$ defines the uncertainty in the location of the object relative to the world coordinate frame, $\mathbf{R}_{\Delta o}$ is the rotational uncertainty about the world Z-axis, and $\mathbf{T}_o$ and $\mathbf{R}_o$ are the ideal location and orientation of the object, respectively. The uncertainty transform for $\mathbf{T}_{\Delta o}$ is

$$\mathbf{T}_{\Delta o} = \begin{pmatrix} 1 & 0 & 0 & \Delta X_o \\ 0 & 1 & 0 & \Delta Y_o \\ 0 & 0 & 1 & \Delta Z_o \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

and the uncertainty matrix for $\mathbf{R}_{\Delta o}$ is

$$\mathbf{R}_{\Delta o} = \begin{pmatrix} \cos(\Delta\theta_o) & -\sin(\Delta\theta_o) & 0 & 0 \\ \sin(\Delta\theta_o) & \cos(\Delta\theta_o) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

where $[\Delta X_o\ \Delta Y_o\ \Delta Z_o\ 1]^t$ and $\Delta\theta_o$ are the dispacement and orientation uncertainty for the object, as shown in Fig. 9.

Other quantities that we must consider are those related to the configuration of the gripper. As shown in Fig. 10, we consider the location of the tool center and the rotation about the Z-axis of the gripper's local frame to be uncertain. Usually, the gripper uncertainty is less than the uncertainty in positions which are determined by the sensing system. The transformation matrix for the gripper uncertainty is given by

$$\mathcal{T}_{\Delta g} = \begin{pmatrix} \cos(\Delta\theta_g) & -\sin(\Delta\theta_g) & 0 & \Delta X_g \\ \sin(\Delta\theta_g) & \cos(\Delta\theta_g) & 0 & \Delta Y_g \\ 0 & 0 & 1 & \Delta Z_g \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{9}$$

where $[\Delta X_g\ \Delta Y_g\ \Delta Z_g]^t$ and $\Delta\theta_g$ are the dispacement and orientation uncertainty variables of the gripper.

IRAS uses a computer-controlled pneumatic clamp for the fixturing operations. Since this fixture is placed on the work table, its Z-position in the world coordinate is known without ambiguity. However, its dispacement about the X- and Y-axis, as well as the rotation about the Z-axis, are not exact. Since the clamp is essentially a stationary parallel-jaw gripper, we want to express the uncertainty of the fixture in the same manner as that of the gripper. In order to do this, we must express the uncertainty of the fixture in terms of its local coordinate, as shown in Fig. 11. Let $\Delta\mathbf{O}_f$ denote the displacement uncertainty of the fixture in the world coordinate. Then its corresponding values in the fixture's local coordinate can be expressed as

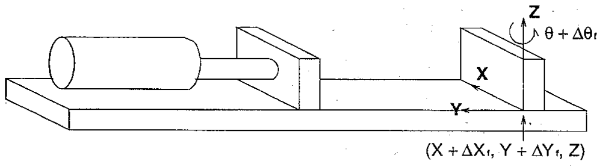


Fig. 11. Possible uncertainties in the configuration of the fixture.

$$[\Delta X_f\ \Delta Y_f\ 0\ 1]^t = \mathbf{R}_f^{-1}\Delta\mathbf{O}_f$$

where $\mathbf{R}_f$ is the ideal orientation of the fixture. Since the Z-axis of both the clamp and the world have the same direction, the rotational uncertainty in the clamp's local coordinate frame is the same as its rotational uncertainty in the world frame. Let this be represented by the uncertainty variable $\Delta\theta_f$. Then the uncertainty transform for the fixture is

$$\mathcal{T}_{\Delta f} = \begin{pmatrix} \cos(\Delta\theta_f) & -\sin(\Delta\theta_f) & 0 & \Delta X_f \\ \sin(\Delta\theta_f) & \cos(\Delta\theta_f) & 0 & \Delta Y_f \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{10}$$

In an uncertain environment, the uncertainties associated with the object descriptions must be within certain bounds if the planned actions are to be assured of success. In this section, we will develop the conditions for the uncertain quantities which ensure that the planned actions can be carried out. The **putdown** action is assumed to always succeed, since IRAS only places objects at locations that are known to be clear on the worktable. Two other actions of IRAS, **unfixture** and **locate**, can always be executed regardless of the uncertainties in the object descriptions. Hence, only the following actions need to be considered: **pickup, fixture,** and **assemble**.

The success of a **pickup** action depends on the possible locations and orientations of both the gripper and the object to be grasped. In particular, the two contact points on the object must lie between the two fingers of the gripper. This is satisfied if the grasp point of the object is located within a certain distance of the tool center of the gripper.

We will now derive the necessary criteria for the success of the pickup action. A pictorial illustration of this is given in Fig. 12. Let $\mathbf{P}_o$ be the grasp point in the object's local coordinate frame. Then its possible value $\mathbf{P}_g$ in the actual gripper coordinate frame is

$$\mathbf{P}_g = \left(\mathcal{T}_w^g\mathcal{T}_{\Delta g}\right)^{-1}\mathcal{T}_w^{o+\Delta}\mathbf{P}_o$$

where $\mathcal{T}_{\Delta g}$ and $\mathcal{T}_w^{o+\Delta}$ are given by (9) and (6), respectively, and $\mathcal{T}_w^g$ is the ideal gripper configuration relative to the world. Let ω_g and ω_o be the width of the gripper opening and the distance between the two contact points on the object, respectively. If the two contact points are to lie between the fingers of the gripper, the Y-component of $\mathbf{P}_g$ must be no

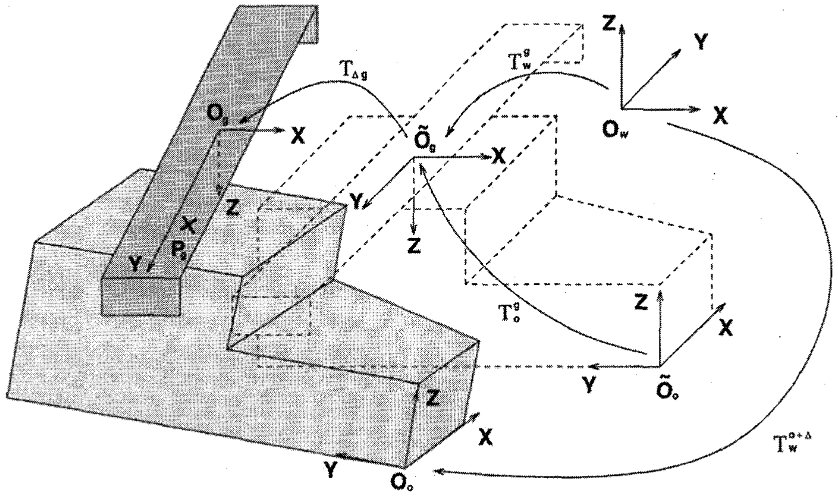


Fig. 12. Possible gripper coordinates and grasp point for a successful pickup action. The symbols $\tilde{\mathbf{O}}_g$ and $\tilde{\mathbf{O}}_o$ represent the ideal local coordinate frames for the gripper and the object, respectively, while $\mathbf{O}_g$ and $\mathbf{O}_o$ denote the actual local coordinate frames for the gripper and the object, respectively.

further than $(\omega_g - \omega_o)/2$ away from $\mathbf{O}_g$, i.e.,

$$-\epsilon < [0\ 1\ 0\ 0]\mathbf{P}_g < \epsilon \tag{11}$$

where $\epsilon = (\omega_g - \omega_o)/2$.

The conditions under which a **fixture** operation will succeed are similar to those of the pickup action. In the fixture operation, the clamps close about the fixture point, whereas in the pickup action the fingers close about the grasp point. In IRAS, the object to be fixtured is held in the gripper just prior to the fixturing operation. Therefore, we need to convert the uncertainty from the gripper coordinate to the fixture coordinate. Let $\mathbf{P}_o$ denote the position of the fixture point in the object's local coordinate. Then the possible locations of this point in the fixture's actual coordinate system is

$$\mathbf{P}_f = (\mathcal{T}_{\Delta f})^{-1}\mathcal{T}_f^o\mathcal{T}_o^g\mathcal{T}_\Delta(\mathcal{T}_o^g)^{-1}\mathbf{P}_o \tag{12}$$

where $\mathcal{T}_{\Delta f}$ is given by 10, $\mathcal{T}_f^o$ is the ideal configuration of the object relative to the fixture, $\mathcal{T}_o^g$ is the ideal grasping configuration, and $\mathcal{T}_\Delta$ is the uncertainty of the object in terms of the gripper coordinate. Using the same argument as those for the pickup action, we get the constraint for the fixture operation

$$-\epsilon < [0\ 1\ 0\ 0]\mathbf{P}_f < \epsilon \tag{13}$$

where $\epsilon = (\omega_g - \omega_o)/2$, and ω_f and ω_o are the width of the clamp opening and the distance between the two contact points on the object, respectively.

The **assemble** action mates the object held in the gripper with another object fixtured with the clamp. In the current implementation, we only account for uncertainties in the assembly operations for mating an object with a round peg to an object with a round hole, as shown in Fig. 13. For mating operations of other kinds of assembly, we assume that the uncertainty tolerance is zero. As will be made clear in Section VIII, this will cause the fine motion planner to be invoked to generate the motions for the assembly process. The peg-in-hole assembly will succeed if all the possible positions of the peg will lie in each of the possible positions of the hole. Let R_p and R_h denote the radius of the peg and the hole, respectively. Then the assembly will succeed only if the relative distance between the center of the peg and the center of the hole is no bigger than $R_h - R_p$, i.e.,

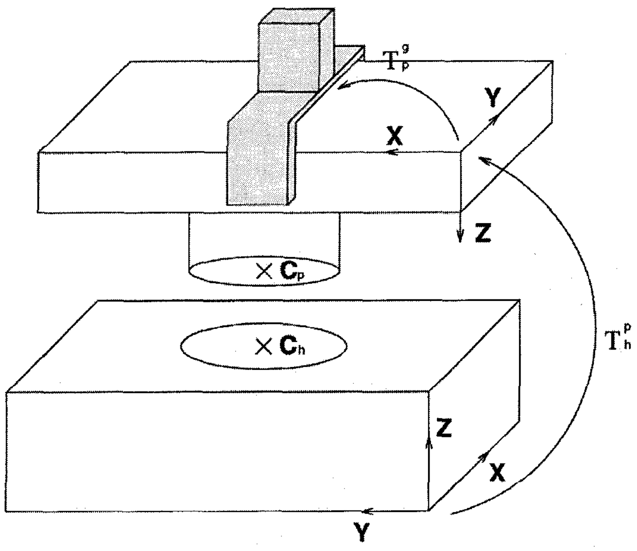


Fig. 13. The relationship between the peg and the hole in the peg-in-hole assembly.

$$R_h - R_p > |\mathbf{C}_h - \mathbf{C}_p| \tag{14}$$

where $\mathbf{C}_h$ and $\mathbf{C}_p$ are the possible locations of the center of the hole and of the peg, respectively, in the local frame of the object with the hole. The value of $\mathbf{C}_h$ can be obtained in a straightforward manner. In IRAS, the object with the hole will be fixtured, so its possible positions in the object's ideal local coordinate frame is

$$\mathbf{C}_h = \mathcal{T}_{\Delta o}\tilde{\mathbf{C}}_h$$

where $\mathcal{T}_{\Delta o}$ is the uncertainty transform of the object relative to the fixture's local coordinate, and $\tilde{\mathbf{C}}_h$ is the center of the hole in the object's local coordinate frame.

The computation for $\mathbf{C}_p$ is less obvious. Since the peg is held by the gripper, its uncertainty will be described in terms of the gripper's local coordinate. Let this uncertainty matrix be represented by $\mathcal{T}_{\Delta}$. Then the possible locations of the center of the peg in the object's local coordinate frame is

$$\mathbf{C} = \mathcal{T}_p^g \mathcal{T}_{\Delta} \left(\mathcal{T}_p^g\right)^{-1} \tilde{\mathbf{C}}_p$$

where $\mathcal{T}_p^g$ is the ideal gripper configuration relative to the peg's local coordinate frame, and $\tilde{\mathbf{C}}_p$ is ideal location of the center of the peg in the object's local coordinate frame. In IRAS, the assembly goal specifies the final position of the peg relative to the position of the object with the hole. Denote this transformation as $\mathcal{T}_h^p$. Then the possible location of the center of the peg in the ideal coordinate frame of the fixtured object is given by

$$\mathbf{C}_p = \mathcal{T}_h^p \mathbf{C}$$

or

$$\mathbf{C}_p = \mathcal{T}_h^p \mathcal{T}_p^g \mathcal{T}_{\Delta} \left(\mathcal{T}_p^g\right)^{-1} \tilde{\mathbf{C}}_p. \tag{15}$$

All of the actions in IRAS change the uncertainty of the object description. Therefore, it is necessary to determine the effects of an action on the uncertainties in the work cell. The six actions of IRAS, **pickup, putdown, fixture, unfixture, locate,** and **assemble**, all affect the uncertainty in different ways. IRAS contains formulas for how uncertainties propagate through these actions. However, these formulas will not be presented here as their veracity has yet to be established through experimentation.

VII. Assembly Task Planning

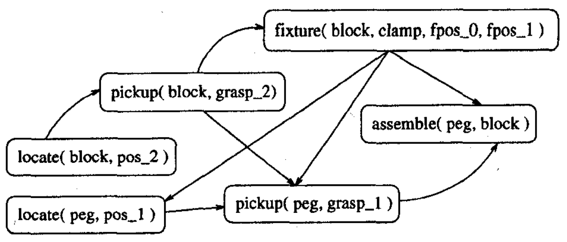
The purpose of the assembly task planning module shown in Fig. 4 is to generate only that part of the overall manipulation plan that is independent of the initial positions and orientations of the parts. The assembly task planner also inserts into the plan sensory requests, which are invoked during execution time for locating the necessary parts for the assembly. Suppose the desired assembly is as displayed in Fig. 2(a) which calls for the assembly of a peg into a block, but the block must first be placed in a fixture in a particular pose. This top level assembly goal is as follows:

```
assembled(peg_in_hole_assembly,
          gripper_held_object(Peg),
          fixtured_object(Block),
          mate_faces(MateFaces),
          mate_transform(MateTM) ,
          mate_vector(MateVector),
          fixture(clamp),
          contact_face_pairs(FacePairs)),
```

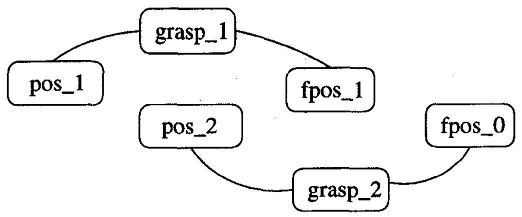
where MateFaces is a list of mating features, MateTM is the transformation that describes the final position and orientation of the peg relative to the block, MateVector specifies the direction of the mating operation, and FacePairs specifies the contact faces between the block and the fixture, which is a clamp in this case. As mentioned earlier, this goal is specified by a human via the DataGlove system. A detailed description of this is given in [32].

Given this assembly goal, the assembly task planner produces the plan shown partially in Fig. 14(a) and an associated constraint network, shown pictorially in Fig. 14(b). As the reader can see, all the variables in this plan are constrained to take on only certain permissible instantiations. The symbols *grasp_1, grasp_2, pos_1, pos_2, fpos_0, fpos_1* are the planning variables; *grasp_1* and *grasp_2* are for picking up the peg and the block, respectively, *pos_1* and *pos_2* will be instantiated to the initial positions and orientations of the peg and block, respectively, *fpos_0* and *fpos_1* will get instantiated to the positions and orientations of the fixtured object before and after the clamps are activated, respectively. In Fig. 14(b), shown below the constraint network are the known permissible instantiations for the planning variables. Therefore, the assembly task planner has already determined that the constraints of the assembly dictate that the instantiations for the variable *grasp_1* be limited to the set {*peg_g1, peg_g13, peg_g17*, ...}, where each of the constant symbols *peg_g1*, etc., actually stands for a particular grasp configuration. The same is the case for the other variables shown in Fig. 14(b). Note also from Fig. 14(b) that while the nodes of the network are constrained in this manner, the arcs themselves are devoid of any constraining information. For example, the arc connecting the nodes marked *grasp_2* and *pos_2* will eventually be constrained by the assembly execution planner using the **reachable** constraint which, as was mentioned earlier, tests for the kinematic feasibility of the robot gripper reaching the object in a certain pose and with a certain grasp for the fixturing operation. If the **reachable** constraint cannot be satisfied, the assembly execution planner will automatically generate regrasping operations, as will be discussed further in the next section.

At this juncture, the reader might ask why this split of the planning process into assembly task planning and assembly execution planning, the former for generating an initial constraint network for the planning variables and the latter for imposing constraints on the arcs of the network and for possibly extending the network. This question is pertinent particularly because in SPAR there is no such division between the two phases of planning. The division of the planning process in IRAS was done for reasons of endowing the system with enhanced reactivity and for computational efficiency. By generating the entire plan all at once, as in SPAR, one runs the risk of having to abandon the whole plan if any of the objects get inadvertently disturbed by the robot. On the other hand, by dividing the planning process in the manner we do in IRAS, one first uses the assembly task planner to create a plan—it may be referred to as the high-level plan—that constrains the grasp, location, and pose variables to those instantiations that would permit the assembly to be executed.

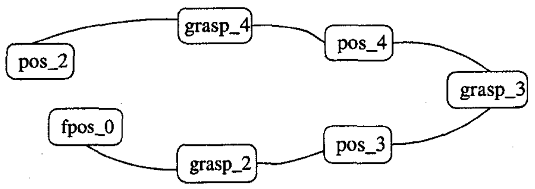


(a)



variable	***its label set***
grasp_1	{peg_g1, peg_g13, peg_g17,...}
grasp_2	{block_g2, block_g9, block_g4,...}
fpose_1	{dbtm(ftm_1)}
fpos_0	{dbtm(tm_1)}

(b)



arc (grasp_4 / pos_2, { block_g18 / dbtm(tm_2) })
arc (grasp_4 / pos_4, { block_g18 / [block_p4,0] })
⋮
arc (grasp_2 / pos_3, { block_g4 / [block_p5,0] })
arc (grasp_2 / fpos_0, { block_g4 / dbtm(tm_1) })

(c)

Fig. 14. The constraint network in (a) and (b) are generated by the assembly task planner. Subsequently, the assembly execution planner augments the constraint network of (b) and generates constraints for the arcs of the network. Shown in (c) is the augmentation of the bottom segment of the network in (b). (a) Prior_to constraints on planned actions. (b) Associated constraint network. (c) Expanded arcs: *grasp_2/pos_2, grasp_2/fpos_0.*

Subsequently, the system uses the assembly execution planner to locate each object, on an as-needed basis, and the calculated location and pose of the object then cause the imposition of kinematic and other constraints on the instantiations permitted by the assembly task planner. More specifically, the assembly execution planner applies kinematic constraints implied by considerations such as reachability, etc., to the nodes of the constraint network. As was shown already in Fig. 14, part (b) of the figure shows the instantiation sets permitted by the assembly task planner at each of the nodes. Fig. 14(c) shows the arc constraints generated by the assembly execution planner for the arc between the nodes *pos_2* and *grasp_2* of Fig. 14(b). Also shown in Fig. 14(c) is the augmentation of the network by the assembly execution planner. This figure will be discussed in greater detail in the following section.

VIII. Assembly Execution Planning

As should already be evident from the discussion in the preceding section, the assembly execution planner has the following functions.

1) Invoke sensing actions for objects as they are needed for the assembly manipulation.
2) Apply reachability and other such criteria to the objects located by the sensors. Invocation of the reachability criteria will generate arc constraints, as shown already in Fig. 14(c).
3) If the object found cannot be reached with the grasp needed for the satisfaction of a subsequent goal, generate a regrasping manipulation sequence. This would cause augmentation of both constraint networks. Notice the assembly execution planner generated augmentation in Fig. 14(c) of the constraint network of Fig. 14(b). This augmentation consists of a regrasping sequence signified by the chain consisting of nodes *grasp_4, pos_4, grasp_3* and *pos_3.* This chain was inserted in place of the link between the nodes *pos_2* and *grasp_2* in the network of Fig. 14(b). That link had to be broken because the assembly execution planner discovered, by invoking kinematic considerations, that no permissible instantiation for the variable *grasp_2* would allow the block to be transferred from *pos_2* to *fpos_0.* The regrasp planning is described in greater detail in the remainder of this section.
4) Examine the partial ordering among the actions, as encoded in the network shown in Fig. 14(a), and then select an action on the basis of the following criteria:
   If an action node in the network has no *prior_to* precedents, it is accorded a high priority for execution.
   If two or more actions have the same priority, that action is selected which is a *prior_to* precedent of the largest number of action nodes. Once an action has been successfully executed, it is deleted from the *prior_to* network. The constraint network is also updated.
5) Apply the uncertainty reasoning process described in Section VI to update the uncertainty in the world description. If the uncertainty level is too large so that success cannot be guaranteed for an action chosen for execution, invoke the appropriate strategies to reduce the uncertainty. Currently, a pair of two distinct grasps are selected to reduce the uncertainty in the position and orientation of an object. In the case of the assembly operation, violation of the uncertainty constraint will be handled with a simple fine-motion planner employing force/torque control [15].

In the remainder of this section, we will describe the method used in IRAS for regrasping and for updating uncertainties.

Regrasping actions are generated by treating the regrasping problem as a discrete one. This is accomplished by quantizing the space of grasp configurations of each object and using stable classes to characterize its placement on the table.

Associated with each stable class is a set of orientations for regrasping, i.e., an instance of a stable pose is specified by a placement-orientation pair. This representation is similar to that described in [21], [30], and [25]. Since regrasping is preceded by the need to put down the object, we have reserved some free space on the table for that purpose.

We construct a grasp-placement graph, $\mathcal{G}$, for an object as follows. Each placement-orientation pair is a vertex in $\mathcal{G}$. $\mathcal{G}$ has two additional vertices, P_0 and P_f, representing the initial and final positions and orientations of the object, respectively. Two vertices in $\mathcal{G}$ are connected together if and only if there is a common grasp that is feasible for the placement corresponding to the vertices, the arc joining the two vertices is labeled by the set of all such grasps.

The construction of $\mathcal{G}$ requires an exhaustive enumeration of pairs of placements and grasps. This is not as difficult as it might seem, since the grasps are characterized by grasping features associated with each stable pose, and the number of stable poses for an object is often fairly small. Furthermore, the subgraph $\mathcal{G}^* = \mathcal{G} - \{P_0, P_f\}$ is the same for each object regardless of its initial and final position and orientation, i.e., $\mathcal{G}^*$ is independent of the task that requires the object. Therefore, we can precompute $\mathcal{G}^*$ and store it in the knowledge base of the system. Once $\mathcal{G}$ has been constructed, the problem of finding a sequence of regrasping operations can be formulated as finding a shortest path from P_f to P_0.

Once an action has been successfully executed, the assembly execution planner updates the uncertainties of the objects affected by the action according to the specifications described in Section VI. Since all of the uncertainty constraints are of the forms $\mathcal{C} < \mathcal{E}$ and $\mathcal{E} < \mathcal{C}$, where $\mathcal{C}$ is a constant and $\mathcal{E}$ is a symbolic expression in terms of uncertainty variables, the evaluation of these constraints can be done in a straightforward manner—for each symbolic expression, it is only necessary to find the lower and upper bounds on the expression, subject to a set of constraints. In IRAS, finding bounds on symbolic expressions is accomplished by using a SUP/INF algorithm. The reader is referred to [2], [4], [5], and [29] for a more detailed description of the SUP/INF method.

IX. Execution Monitoring and Sensing

The execution-monitoring system is devised for a work cell equipped with a PUMA762 robot manipulator. The sensors consist of a parallel-jaw gripper (used as a sensor to measure the width of the part of the object that is grasped), a wrist-mounted force/torque sensor, as well as various kinds of cameras. As discussed presently, the cameras are used to determine the positions and orientations of the part specified in a sensory request. The sensor system monitors the execution of each action to ensure its success. In the case of a pickup action, the width of the opening of the gripper is used to determine the success of the operation. If this action cannot be executed successfully, perhaps because the part was located too close to other parts, the assembly execution planner is informed of the failure so that it can issue a new sensory request to locate the part and replan the regrasping steps. For the assemble action, the force/torque sensor is used to ensure the success of the mating operation. The method described in [15] is employed to recover from mating errors. Other operations such as putdown and fixturing objects are assumed to always succeed.

Regarding the vision part of sensing, it is important to note that in light of the great strides that have been made in 3-D vision [9], [16], [19], [17], [20], [11], [13], it would certainly be possible today to endow IRAS with a sophisticated and robust vision module for recognizing and locating the objects needed for assembly. However, the focus of this research is more on planning than on vision. So, for the purpose of establishing the feasibility of the planning notions presented here, we developed a special 2-D vision module for IRAS that is simple and fast. This module is based on the assumption that the objects are recognizable from their 2-D bounding contours, the objects resting in arbitrary stable poses on a flat surface. This module uses two cameras. One of the cameras, mounted overhead so that it can view the entire work area, is used for coarse location of all the objects in the work area. The other camera is mountable in the gripper and is used as needed for the determination of the positions and orientations of the parts. The gripper-held camera is oriented so that its image plane is parallel to the worktable. For an object in one of its stable configurations, the image taken by the camera will contain the bounding contour of the object that can then be used for identifying the pose. The bounding contour is represented by a differential chain code, which is refined using a least square fit method to yield analytic forms. From the bounding contour, the stable pose of the object is recognized using a minimum least squares classifier. For an object in a stable pose, it only has one rotational degree of freedom—a rotation about a direction perpendicular to the table. Thus, the identification of this stable pose in conjunction with the angle of rotation specifies the orientation of the object. This angle, along with the displacement of the object, is computed using the contour of the object. Fiducial marks are placed on the different faces of the objects; these marks help determine the location of the origin of the local coordinate frame attached to the object. To illustrate, the parts used for the assembly shown in Fig. 3 are marked with triangular markers, as illustrated in Fig. 15.

Regarding the processing of the overhead camera image, a histrogram derived threshold is applied to the image and the resulting components labeled. Since the objects to be manipulated are not very large, the image provided by the overhead camera does not provide enough data that can be used to differentiate the parts. But we can compute the center of mass of each component, that after application of the inverse perspective transform for the camera yields the world coordinates of the center of mass of the object. The inverse perspective transform is performed using the two-plane method of camera calibration [24].

Once a rough estimate of the object's location is available, the robot moves its gripper-held camera so that it is, at a predesignated height, directly above the center of the object as ascertained from the overhead camera image. The camera is oriented such that its image plane is parallel to the table. With this arrangement, the entire object is within the field of view of the camera. Because the object is in a stable configuration, the image taken by the camera will contain the bounding contour

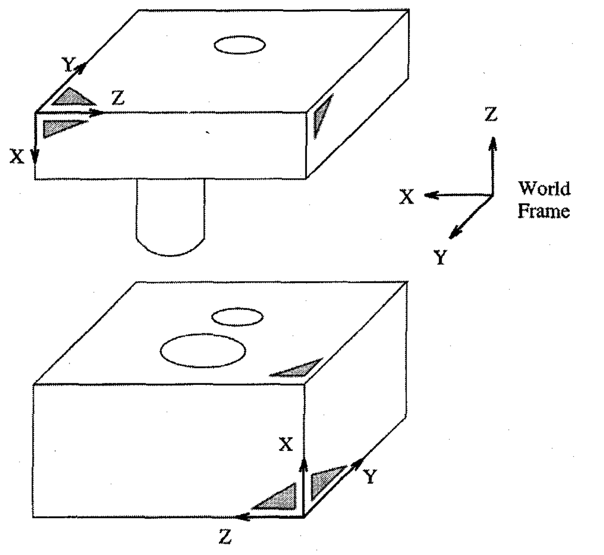


Fig. 15. The fiducial markers for the parts for the sample assembly shown in Fig. 3.

of the object, which is used to identify and compute the pose of the object. This process is carried out in three steps: image segmentation, feature extraction, and pose estimation.

First, the image is thresholded and components labeled. This uses the same component labeling algorithm as for the overhead camera image. In our experiments, because of the camera placement, the largest component in the image will always correspond to the object of interest. Once the image has been labeled, feature extraction can be performed. The system extracts five features from the image: the perimeter of the object, the area of the object, the number of corners of the object, the number of holes in the object, and the location of the fiducial mark.[2] The perimeter of an object is defined to be the total number of border pixels of the object. The area of the object is defined to be the total number of pixels of the object. This is computed by the image segmentation routines described above. The number of corners refers to the number of corners in the border of the object. These five features form the feature space which is used to identify the object and its stable pose.

Initially, each straight line segment is represented by a differential chain code. Subsequently, linear regression is used to fit the standard analytic form $y = a + bx$ to each straight segment and the values of the parameters a and b computed for the segment. Simple formulas then yield the precise locations of the intersections of these straight segments. Finally, in order to locate the fiducial marks, search is conducted along a bisector of the angle formed at each intersection point. In order to determine whether or not an object face contains a hole, a search is conducted along the two principal axes in the vicinity of the center of the object.

Of the five features mentioned above—area, perimeter, number of corners in the contour, number of holes, and location of the fiducial mark—the first four are used for constructing a 4-D feature space that is used for distinguishing between the different stable pose classes of an object, each stable pose class characterized by the object face in contact with the table. Each stable pose class is represented by an examplar and the classification consists of selecting that pose class which yields the minimum least squares error for the distance between the measured features and the exemplar. The exemplar for each stable pose class is selected by simply averaging the feature values corresponding to the many instances of that stable pose class shown to the camera system during a training session.

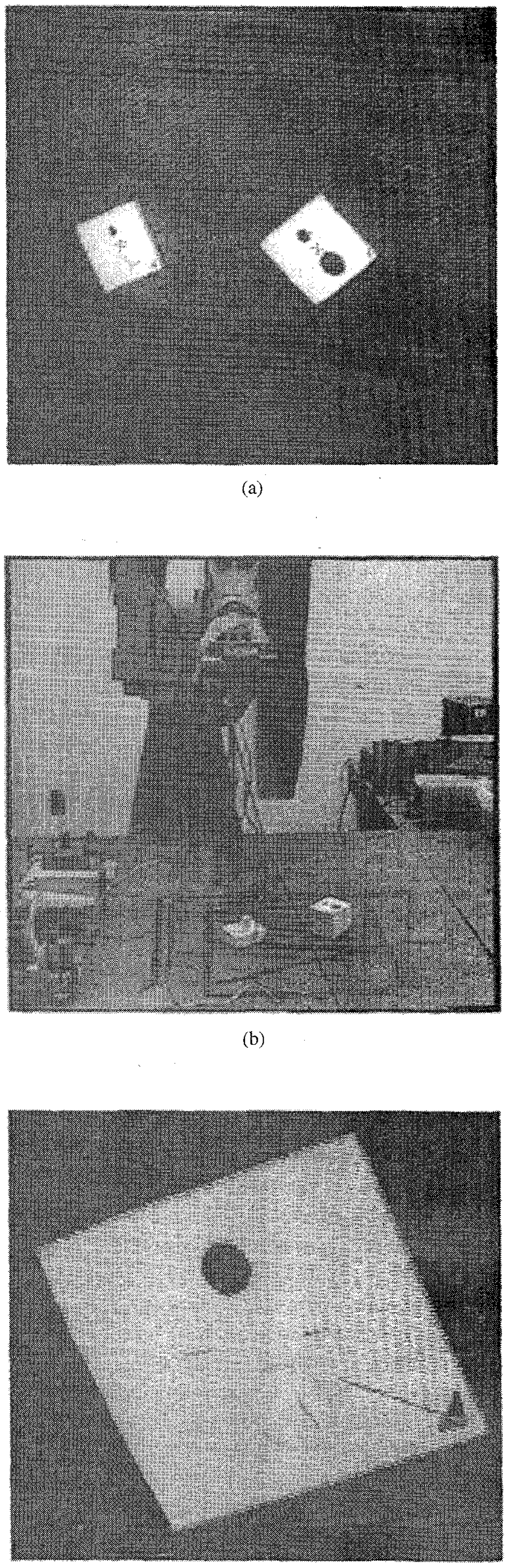
Once the stable pose class of an object on the table is determined, its precise pose is computed from the location of the fiducial mark. Note that in each stable pose class, the only degree of freedom in the object pose is its rotation about the world Z-axis, which is perpendicular to the worktable. The rotation of the object around this axis (or, more precisely, the rotation of the local object-centered coordinate frame with respect to the world Z-axis) is computed using the two edges adjacent to the fiducial mark associated with this stable pose.

A pictorial illustration of this algorithm is given in Fig. 16. This figure consists of three snapshots taken during an assembly session similar to that of Fig. 3. In particular, it depicts the steps for finding the position and orientation of the peg object. Fig. 16(a) shows the image obtained with the overhead camera. The centroids of the objects are drawn as crosses and superimposed over the image. These values are converted into their respective world coordinates. The gripper-held camera is then placed directly over the object to be classified, as shown in Fig. 16(b). The image obtained with the gripper-held camera is subsequently analyzed in the manner described above. Fig. 16(c) shows the image of the peg object on which are superimposed the extracted boundary and the principle axes of the object face visible to the camera. The diagonal line in the lower right corner illustrates the search direction for finding the fiducial marker.

X. Conclusion

In this paper, we presented IRAS, an implemented robotic assembly system where planning, sensing, and task execution are tightly integrated as interleaved operations. The planning process divides very naturally into two phases. In the first phase, which we term the "assembly task planning" phase, a nonlinear planner is invoked to create a family of general plans that is independent of the dynamic aspects of the work cell. The second phase, which we call the "assembly execution planning" phase, is intertwined with the execution of the plan—the module issues sensory requests, adds detailed steps to the plan in accordance with the current status of the work cell and schedules them for execution. The plans generated by these planners are guaranteed to be correct, since the assembly task planner guarantees that the plans it generates are correct [8], and the steps added by the assembly execution planner are themselves correct and do not clobber planned steps generated in the first phase. Also, the efficient handling of both geometric and kinematic constraints greatly reduces the amount of search during plan generation. Efficient

[2]Note: Location of the fiducial mark refers to the location of the corner nearest to the fiducial mark.


(a)

(b)

(c)

Fig. 16. Locating the peg using the overhead and the gripper-held cameras. (a) Centroids superimposed on the regions extracted from the image from overhead camera. (b) Placing the hand-heald camera over the peg. (c) Orientations and boundary of peg.

uncertainty propagation and constraint evaluation improves the system's ability to detect and to recover from potential errors. Furthermore, the system's ability to plan for sensory requests enhances both the robustness and flexibility of the system. Since only a rudimentary vision module is plugged into the system at this time, the experiment shown in Fig. 3 works all the time as long as the parts thrown on the table form nonoverlapping images in the overhead camera and as long as these parts lie within a prescribed portion of the robot workspace.

APPENDIX
DESCRIPTION OF FIG. 3

For interested readers, here is a brief description of what is being depicted by the different frames of Fig. 3.

Shown in frame Fig. 3(a) is a human throwing the parts in front of the robot. The photo in Fig. 3(b) shows the robot using a gripper-held camera for the localization and pose calculation of the parts in the work area. Since only 2-D vision is used, the different surfaces of the parts are marked with fiducials to aid in pose calculation. The camera is removed subsequently. The sequence of frames in Fig. 3(c) through (f) demonstrates the robot employing two grasping and regrasping operations that are needed to place the block in a pose that then permit the robot to put it down in the fixture, as shown by the pickup operation in Fig. 3(g) and fixturing of the block in Fig. 3(h).

The robot then goes about the business of locating the second part, the peg, needed for the assembly. The robot has the freedom to either use the overhead camera, not shown, or request that the human provide it again with the gripper-held camera for this purpose. In the particular experiment depicted here, an extra grasping operation is needed for the peg in order to make the assembly feasible; the pickup and putdown corresponding to this step are shown in Fig. 3(i) and (j). Shown in Fig. 3(k) is the final pickup of the peg prior to its assembly with the block that is already in the fixture. The assembly itself is shown in the frame Fig. 3(l).

REFERENCES

[1] J. Allen, J. Hendler, and A. Tate, *Readings in Planning*. Los Altos, CA: Morgan Kaufmann, 1990.

[2] W. W. Bledsoe, "The SUP-INF method in Presburger arithmetic," Dep. Math., Univ. Texas, Austin, Memo. ATP-18, Dec. 1974.

[3] R. P. Brent, *Algorithms for Minimization Without Derivatives*. Englewood Cliffs, NJ: Prentice-Hall, 1973.

[4] R. A. Brooks, "Symbolic reasoning among 3D models and 2D images," *Artificial Intell.*, vol. 17, pp. 285–348, 1981.

[5] ——, "Symbolic error analysis and robot planning," *Int. J. Robot. Res.*, vol. 1, no. 4, pp. 29–68, 1982.

[6] ——, "A robust layered control system for a mobile robot," *IEEE Trans. Robot. Automat.*, vol. RA-2, no. 1, pp. 14–23, 1986.

[7] ——, "A robot that walks: Emergent behaviors from a carefully evolved network," in *Proc. AAAI-86*, 1986, pp. 626–631.

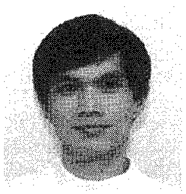
[8] D. Chapman, "Planning for conjunctive goals," *Artificial Intell.*, vol. 32, pp. 333–377, 1987.

[9] C. H. Chen and A. C. Kak, "A robot vision system for recognizing 3-D objects in low-order polynomial time," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, pp. 1535–1563, 1989.

[10] L. S. Hommem de Mello and A. C. Sanderson, "A correct and complete algorithm for the generation of mechanical assembly sequences," *IEEE Trans. Robot. Automat.*, vol. 7, no. 2, pp. 228–240, 1991.

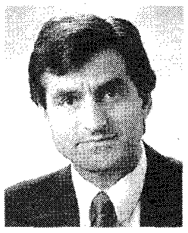
[11] O. D. Faugeras and M. Hebert, "The representation, recognition, and locating of 3D objects," *Int. J. Robot. Res.*, vol. 5, pp. 27–52, 1986.

[12] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial Intell.*, vol. 2, pp. 189–208, 1971.
[13] P. Flynn and A. Jain, "3D object recognition using invariant feature indexing of interpretation tables," *CVGIP: Image Understanding*, vol. 55, no. 2, pp. 119–129, 1992.
[14] B. R. Fox and K. G. Kempf, "Opportunistic scheduling for robotic assembly," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1985, pp. 880–889.
[15] S. N. Gottschlich and A. C. Kak, "A dynamic approach to high-precision parts mating," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, pp. 797–810, Aug. 1989.
[16] L. L. Grewe and A. C. Kak, "Interactive learning of a multiple-attribute hash table classifier for fast 3D object recognition," *Comput. Vision Image Understanding* (formerly *CVGIP-IU*), vol. 61, pp. 387–416, May 1995.
[17] W. E. L. Grimson and T. Lozano-Perez, "Model-based recognition and localization from sparse range or tactile data," *Int. J. Robot. Res.*, vol. 3, pp. 3–35, 1987.
[18] S. A. Hutchinson and A. C. Kak, "SPAR: A planner that satisfies operational and geometric goals in uncertain environments," *AI Mag.*, vol. 11, pp. 30–61, Spring 1990.
[19] K. Ikeuchi, "Generating an interpretation tree from a CAD model for 3D-object recognition in bin-picking tasks," *Int. J. Comput. Vision*, vol. 1, pp. 145–165, 1987.
[20] W. Y. Kim and A. C. Kak, "3D object recognition using bipartite matching embedded in discrete relaxation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, no. 3, pp. 224–251, 1991.
[21] T. Lozano-Perez, J. L. Jones, E. Mazer, P. A. O'Donnell, W. E. L. Grimson, P. Tournasoud, and A. Lanusse, "Handey: A robot system that recognizes, plans, and manipulates," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1987, pp. 843–849.
[22] D. M. Lyons, A. J. Hendriks, and S. Mehta, "Achieving robustness by casting planning as adaptation of a reactive system," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1991, pp. 198–203.
[23] D. M. Lyons and A. J. Hendriks, "Planning for reactive robot behavior," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1992, pp. 2675–2689.
[24] H. A. Martins, J. R. Birk, and R. B. Kelly, "Camera models based on data from two calibration planes," *Comput. Vision, Graphics Image Processing*, vol. 17, pp. 173–180, 1981.
[25] J. Pertin-Troccaz, "On-line automatic robot programming: A case study in grasping," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1987, pp. 1292–1297.
[26] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipies in C: The Art of Scientific Computing*. Cambridge: Cambridge Univ. Press, 1991.
[27] C. Ramos and E. Oliveira, "Closing the loop of task planning, action and sensing," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 1992, pp. 909–916.
[28] A. C. Sanderson, H. Zhang, and L. S. Hommem de Mello, "Assembly sequence planning," *AI Mag.*, vol. 11, no. 1, pp. 62–81, Spring 1990.
[29] R. E. Shostak, "On the SUP-INF method for proving Presburger formulas," *J. ACM*, vol. 24, no. 4, pp. 529–543, 1977.
[30] P. Tournassoud and T. Lozano-Perez, "Regrasping," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1987, pp. 1924–1928.
[31] C.-P. Tung and A. C. Kak, "Integrating sensing, task planning and execution," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1994, pp. 2030–2037.
[32] ——, "Automatic learning of assembly tasks using a DataGlove system," in *Proc. IROS '95*, Pittsburgh, PA, pp. 1–8.
[33] X. Xia and G. A. Bekey, "SROMA: An adaptive scheduler for robotic assembly systems," in *Proc. IEEE Int. Conf. Robotics and Automation*, 1988, pp. 1282–1287.



Chao-Ping Tung (S'93–M'95) received the B.S., M.S., and Ph.D. degrees in 1988, 1989, and 1995, respectively, all from Purdue University, West Lafayette, IN.

He was a Research Associate at the Robot Vision Laboratory at Purdue from 1990 to the spring of 1995, working in the areas of sensor-based robotics, computer vision, and artificial intelligence. He is currently a Senior Engineer at NEC America, Inc.



Avinash C. Kak (M'71) is currently serving a two-year term as an IEEE Distinguised Lecturer in the Robotics and Automation Area. He is a coauthor of the widely used *Digital Picture Processing* (New York: Academic, 1982) and a coauthor of a forthcoming book entitled *Robotic Planning*.