MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo No. 842 April, 1985

An Approach to Automatic Robot Programming

Tomás Lozano-Pérez
Rodney A. Brooks

Abstract. In this paper we propose an architecture for a new task-level system, which we call TWAIN. *Task-level* programming attempts to simplify the robot programming process by requiring that the user specify only goals for the physical relationships among objects, rather than the motions needed to achieve those goals. A task-level specification is meant to be completely robot independent; no positions or paths that depend on the robot geometry or kinematics are specified by the user. We have two goals for this paper. The first is to present a more unified treatment of some individual pieces of research in task planning, whose relationship has not previously been described. The second is to provide a new framework for further research in task-planning. This is a slightly modified version of a paper that appeared in Proceedings of *Solid Modeling by Computers: from Theory to Applications, Research laboratories Symposium Series,* sponsored by General Motors, Warren, Michigan, September 1983.

Acknowledgements. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence research is provided in part by the Office of Naval Research under Office of Naval Research contract N00014-81-K-0494 and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80-C-0505 and N00014-82-K-0334.

© Massachusetts Institute of Technology

1. Introduction

One of the earliest and most elusive goals of robotics has been the ability to program manipulators at the level of task operations rather than that of individual motions. What at first appeared to be a relatively simple problem was soon discovered to have unsuspected depths. As a result, several proposals to develop such a *task-level* robot programming system have not culminated in a working system. Nevertheless attempts to implement the proposals have led to crisper problem statements, better algorithms and, above all, they have provided a useful framework for research. Over the past six or seven years, and partly in response to difficulties encountered implementing these proposals, significant advances have been made in the theory and practice of task-level programming. As a result of new developments, the architecture of the proposed systems is no longer an adequate framework for research.

In this paper we propose an architecture for a new task-level system, which we call TWAIN. In contrast to earlier proposals, much of the theoretical underpinning of TWAIN exists, and many of the components have been implemented and tested as stand-alone systems in our laboratory. Although these components require additional work, we believe they provide a solid basis for a new effort at integration. We have two goals for this work. The first is to integrate some individual pieces of research in task planning, whose relationship has not previously been explored. The second is to provide a framework for further research in task-planning.

In this section, we provide an overview of task-level programming. Section 2 outlines the architecture of TWAIN. Section 3 through 5 describe, with the TWAIN framework, approaches to the key task-planning modules of TWAIN.

1.1. Levels of Robot Programming

Most robots are programmed manually by moving them through a sequence of desired positions, recording the internal joint coordinates corresponding to each position, and then recording the operations, such as closing a gripper or activating a welding gun, at those positions. The resulting program is a sequence of vectors of joint coordinates and activation signals for external equipment. These programs are executed by moving the robot through the specified sequence of joint coordinates and issuing the indicated signals. This method of robot programming is known as *teaching by showing*, or *guiding*.

Guiding is simple to use and to implement but subject to important limitations, particularly in the use of sensors. Because guiding specifies a single execution sequence for the robot, there can be no loops, conditionals, or computations. In some applications, such as spot welding, painting, and the handling of simple materials, this is enough. To do other applications, including mechanical assembly and inspection, the robot must respond to sensory input, data retrieval, or computation. This type of programming requires the capabilities of a general-purpose computer programming language.

Robot-level languages are traditional computer programming languages that have been extended with commands to access sensors and to specify robot motions. Data from external sensors, such as vision and force, may be used to modify the robot's motions, enabling the robot to operate with greater uncertainty in the position of external objects, thereby increasing their range of application. The key disadvantage of robot-level programming languages, relative to guiding, is that they require the robot programmer to be expert in computer programming and in the design of sensor-based motion strategies.

Task-level programming simplifies the robot programming process by requiring that the user specify goals for the physical relationships among objects, rather than the motions of the robot needed to achieve those goals. The task-level specification is robot independent, in that no position or path that depends on the robot geometry or kinematics is specified by the user. Because task-level programming systems require complete geometric models of the environment and of the robot as input, they are also referred to as world-modeling systems.

Task-level programming is one of the earliest and most elusive goals of Robotics Science. What at first appeared to be a relatively simple problem was soon discovered to have unsuspected depths. As a result, several attempts to develop task-level robot programming systems have not culminated in working systems. Nevertheless, attempts to implement the proposed systems have led to crisper problem statements, better algorithms, and useful frameworks for research. It is partly a result of these signifcant advances in the theory and practice of task-level robot programming that we propose to develop a new task-level system.

1.2. Basic problems in task planning

In task-level programming, the task planner converts the user's specification of a task into a robot-level program to carry out the task[1]. The main role of the task planner is to plan the robot-specific motion and sensing commands necessary to achieve the task.

Consider a simple block-stacking example (figure 1). The task can be specified as follows:

```
PLACE A SUCH THAT (A.4 AGAINST TABLE) AND
                  (A.1 AGAINST F.1) AND (A.2 AGAINST F.2)
PLACE B SUCH THAT (B.4 AGAINST A.3) AND
                  (A.1 COPLANAR B.1) AND (A.2 COPLANAR B.2)
```

In the absence of positioning errors in the manipulator or in our knowledge of the position of parts, this task could be accomplished with the following program:

[1]We assume that the input description of the task completely specifies the sequence of assembly. Alternatively, another program called an *assembly planner* produces such a description from the user's description.

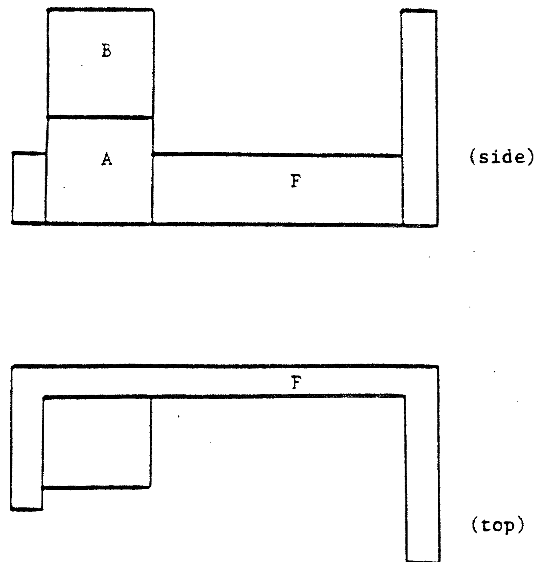

Figure 1. A block-stacking example.

```
1.  OPEN-FINGERS TO <width of A + epsilon>
2.  MOVE TO <location of grasp on A> VIA <path>
3.  CLOSE-FINGERS TO <width of A - epsilon>
4.  MOVE TO <location at F> VIA <path>
5.  OPEN-FINGERS TO <width of B + epsilon>
6.  MOVE TO <location of grasp on B> VIA <path>
7.  CLOSE-FINGERS TO <width of B - epsilon>
8.  MOVE TO <location on A> VIA <path>
9.  OPEN-FINGERS TO <width of B + epsilon>
```

To generate this program, the task planner chooses positions for objects initially in the workspace, determines how new objects will be fed into the workspace, picks grasp points on parts, and finds paths that avoid collisions. These problems are not the only ones that face a task planner. In practice, this simple program fails to achieve the desired goals because of the presence of errors in the robot control system and in the system's knowledge of the location of the parts. The use of sensing can, in many cases, overcome these problems, but this requires that the task planner decide what kind of sensing is useful and how to combine it with the appropriate motions. In fact, dealing with uncertainty permeates all of task planning.

In summary, to convert a task-level specification to a robot-level specification, a realistic task planner must solve the following problems:

- *Parts feeding* — The planner must choose how to introduce each of the parts required for the assembly into the workspace in ways that maximize speed and reliability in acquiring the parts. In the example in figure 1, the type of feeder for A and B must be determined.
- *Layout* — The planner must choose where in the workspace each operation

is to take place in ways that minimize sensing and positioning error as well as reduce the time for the complete assembly. In the example in figure 1, the location of F and of the feeders for A and B must be chosen.

- *Fixturing* — The planner must choose jigs and fixtures to hold the parts to the required accuracy under the forces generated by assembly motions. In the example in figure 1, F serves the function of a fixture; no further fixturing is needed.
- *Fine motion* — The planner must choose a strategy of sensing and motion that guarantees that parts-mating operations will be reliable despite errors in control and sensing. In the example in figure 1, strategies must be chosen that guarantee that A reaches the corner of F and that B is aligned with A.
- *Grasping* — The planner must choose how to grasp each part so the grasp is stable, avoiding collisions while grasping or while placing the part at its destination. In the example, the planner must ensure that the grasp points on A and B are stable, enabling the assembly operations to take place, and that the grasping motions do not introduce too much error in position.
- *Gross motion* — The planner must choose efficient collision-free paths for the manipulator and the parts it carries. In the example in figure 1, the path taking A from the feeder to near F and taking B from the feeder to above A is chosen to avoid collisions of the manipulator with F and the feeder (even in the presence of position control error).

One possible program for the block-stacking task (taking into account the likelihood of errors) has the following structure:

```
1.  OPEN-FINGERS TO <width>
2.  MOVE TO <location of grasp on A> VIA <path>
3.  CLOSE-FINGERS TO <width>
4.  MOVE TO <approach location near F> VIA <path>
5.  COMPLIANT-MOVE ALONG <direction> UNTIL <contact with F.1 & F.2>
6.  COMPLIANT-MOVE ALONG <direction> UNTIL <contact with TABLE>
7.  OPEN-FINGERS TO <width>
8.  VISION-LOCATE B NEAR <expected location of B>
9.  OPEN-FINGERS TO <width>
10. MOVE TO <location of grasp on B> VIA <path>
11. CLOSE-FINGERS TO <width>
12. VISION-LOCATE B NEAR <expected location of B in fingers>
13. MOVE TO <approach location near B> VIA <path>
14. COMPLIANT-MOVE ALONG <direction> UNTIL <contact with A.3>
```

Note that a simple task-level description leads to a complex robot-level program because of the presence of positioning errors and uncertainty. In fact, this program is inadequate for the task because it does not take into account the likelihood that some operation will fail due to unexpected events. No program ever handles

all possible eventualities, but the addition of sensing significantly increases the reliability of the operation, usually at the expense of speed. This example points out the difficulty of robot-level programming and the potential value of task-level robot programming.

Many of the decisions that had to be made in synthesizing the example program are not obvious from a first glance at the program. These complex decisions manifest themselves only in the numeric values of positions, that is, in the positions of feeders and grasp points, the width of finger openings, directions of compliant motions, and the paths for positioning motions. The basic program structure itself — the need for the compliant motions in steps 5 and 6 and the sensing operation at steps 8 and 12 — is based on numerical estimates of errors in sensing and control (partly based on the result of previous decisions). These decisions are tightly inter-related and propagate across what, on the surface, appear to be independent operations. The choice of grasp points affects the assembly operations, the need for sensing, and the choice of paths.

The design of a planner capable of transforming a task-level specification into a detailed robot program is complex. Our approach to the design is based on a set of key ideas and methods:

- The use of a small number of powerful planning modules to identify the range of possible values of the parameters needed for grasping, gross motion, and fine motion.
- The use constraint propagation to choose feasible values for parameters that affect more than one operation.
- The use of skeleton programs to indicate stereotyped sequences of operations required to execute common tasks.
- The use of configuration space to reason about legal robot motions for grasping, fine motion, and gross motion.

1.3. Hierarchical decomposition and robot programming

Decomposition into independent sub-problems, especially hierarchical decomposition, is one of the most common and powerful of conceptual tools for problem-solving. It is no surprise that it forms the foundation of almost all programming methodologies. Structured programming, for example, is essentially an endorsement of hierarchical decomposition in programs. More recent developments in programming languages, such as data encapsulation, apply this approach for data structures. Of course, even the simplest programming tasks are not completely decomposable; side-effects, such as modification of databases, propagate dependencies across operations. The driving force behind most programming methodologies is to minimize these dependencies.

It is natural to attempt to apply methodologies based on hierarchical decomposition to robot programming and control. However, the fundamental character of robot operations limits the scope of such decomposition. The key difficulties are error and geometry, both of which are non-local. The choice of

grasp point on a part, for example, determines what motions of the hand are required to position the part. The choice of grasp point, in turn, is determined by subsequent assembly operations; for example, a finger cannot be on a surface that is to be against another surface. Nor can the axis between the fingers be perpendicular to large applied forces because only friction is holding the object along that direction. Similarly, the existence of a path to a destination might depend on how the part is held in the hand. Finally, each operation makes certain assumptions about accuracy in locations and shape which affect subsequent operations. These dependencies conspire to make robot operations appear monolithic; one often concludes that everything must be decided before anything at all is decided.

The use of sensing and compliance increases the class of situations in which particular operations will succeed and, thereby, reduces inter-dependence among operations. In the extreme case, each operation can sense the complete current environment and decide independently which method is adequate to perform its task. This is extremely wasteful even assuming that the sensing is free; it will require frequent re-grasping, for example. In general, the low-level decisions made in carrying out each task-level step (layout, grasping, paths, and sensing) influence the decisions that need to be made for steps before and after this one [Brooks 82, Taylor 76]. Because of this, TWAIN is based on a two-level approach to task planning based on propagating constraints on physical quantities.

The TWAIN approach to robot program synthesis starts by expanding the task-level description into a skeleton program. This skeleton program makes reference to quantities provided in the input, quantities that must be chosen by the system, and error quantities on which only bounds are available. Symbolic algebraic constraints are used to make explicit the interactions among the quantities across program steps. The goal of the system is to make few arbitrary decisions, but instead to exploit the mutual constraints among steps to force decisions. Each planning module restricts the values of variables in the skeleton plan in addition to determining the actual operations needed to achieve a step in the program. The planner propagates the restrictions on variables across operations: forwards from constraints on input quantities to constraints on output quantities and backwards from constraints on output quantities to constraints on input quantities [Brooks 82].

1.4. Modeling Requirements

Task-level planners require a complete world model and a complete task specification.

1.4.1. The World Model

The legal motions of an object are constrained by the presence of other objects in the environment, and the form of the constraints depends in detail on the shapes of the objects. Therefore, a task planner needs complete geometric descriptions of objects. There are additional constraints on motion imposed by the kinematic structure of the robot itself. If the robot is turning a crank or opening a valve, then the kinematics of the crank and the valve impose additional restrictions

on the robot's motion. The kinematic models provide the task planner with the information required to plan manipulator motions that are consistent with external constraints. Note that as a result of the robot's operation, new linkages may be created and old linkages destroyed; the task-planner must be appraised of the changes.

In planning robot operations, many of the physical characteristics of objects play important roles. The mass and inertia of parts, for example, determine how fast they can be moved or how much force can be applied to them before they fall over. Similarly, the coefficient of friction between a peg and a hole affects the jamming conditions during insertion. Likewise the physical constants of the robot links are used in the dynamics computation and in the control of the robot.

The feasible operations of a robot are not sufficiently characterized by its geometrical, kinematical, and physical descriptions. One important additional aspect of a robot system is its sensing capabilities of touch, force, and vision. For task-planning purposes, vision enables obtaining the configuration of an object to some specified accuracy at execution time; force sensing allows the use of compliant motions; touch information serves in both capacities. In addition to sensing, there are many individual characteristics of manipulators that must be described; velocity and acceleration bounds, positioning accuracy of each of the joints, and workspace bounds are examples.

1.4.2. The Task Model

A model state is given by the configurations of all the objects in the environment; tasks are actually defined by sequences of states of the world model or transformations of the states. The level of detail in the sequence needed to specify a task depends on the capabilities of the task planner.

The configurations of objects needed to specify a model state can be provided explicitly, as offsets and Euler angles for rigid bodies and as joint parameters for linkages, but this type of specification is cumbersome and error prone. Three alternative methods for specifying configurations have been developed:

- Use a light–pen to position CAD models of the objects at the desired configurations.
- Use the robot itself to specify robot configurations and to locate features of the objects [Grossman and Taylor 78].
- Use symbolic spatial relationships among object features to constrain the configurations of objects, as in $Face_1$ $AGAINST$ $Face_2$ [Popplestone, Ambler, and Bellos 78].

One advantage of using symbolic spatial relationships is that the configurations they denote are not limited to the accuracy of a light–pen or a manipulator. Another advantage of this method is that families of configurations, such as those on a surface or along an edge, can be expressed. Inasmuch as these relationships are easy to interpret by a human, they are easy to specify and modify. The principal disadvantage of using symbolic spatial relationships is that they do not specify

configurations directly; they must be converted into numbers or equations before they can be used.

Recall that model states are simply sets of configurations. If task specifications were simply sequences of models, then given a method such as symbolic spatial relationships for specifying configurations, we should be able to specify tasks. This approach has several important limitations. One limitation is that a set of configurations may overspecify a state, for example, a round pin in a round hole. This problem can be solved by treating the symbolic spatial relationships themselves as specifying the state, since these relationships can express families of configurations. A more fundamental limitation is that geometric and kinematic models of an operation's final state are not always a complete specification of the desired operation. One example of this is the need to specify how hard to tighten a bolt during an assembly. In general, a complete description of a task may need to include parameters of the operations used to reach one task state from another.

The alternative to task specification by a sequence of model states is specification by a sequence of operations. Thus, instead of building a model of an object in its desired configuration, we can describe the operation by which it can be achieved. The description should still be object-oriented, not robot-oriented; for example, the target torque for tightening a bolt should be specified relative to the bolt and not relative to the manipulator. Most operations also include a goal statement involving spatial relationships between objects. The spatial relationships given in the goal specification not only describe configurations; they indicate the physical relationships between objects that should be achieved by the operation. When we say that two surfaces should be *AGAINST* each other, for example, the robot should perform a compliant motion that moves until the contact between the surfaces is actually detected. This is quite different from computing the position where the contact should have occurred — assuming perfect knowledge – and commanding the robot to move to that position. For these reasons, existing proposals for task-level programming languages have adopted an operation-centered approach to task specification [Lieberman and Wesley 77, Lozano-Pérez 76]. TWAIN also assumes this form of input.

1.5. Previous work

A number of task-level language systems have been proposed, but no complete system has been implemented. In this section we briefly review these systems.

The Stanford `Hand-Eye` system [Feldman, et al. 71] was the first of the task-level system proposals. A subset of this proposal was implemented [Paul 72] in a program called `Move-Instance` that chose stable grasping positions on polyhedra and planned a motion to approach and move the object. The planning did not involve obstacle avoidance (except for the table surface) or the planning of sensory operations.

`AL` [Finkel, et al. 74], as initially defined, called for the ability to specify models in `AL` and to allow specification of operations in terms of these models. This has been the subject of some research [Binford 79, Taylor 76], but the results have not

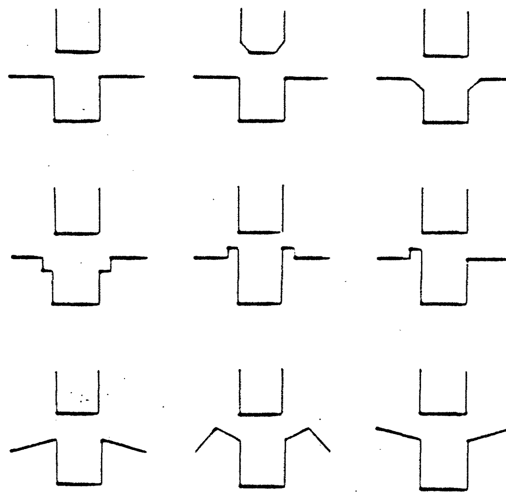

Figure 2. Similar peg-in-hole tasks that require different strategies.

been incorporated into the existing AL system. Some additional work within the context of Stanford's Acronym system [Brooks 81] has dealt with planning grasp positions [Binford 79], but AL has been viewed as the target language rather than the user language.

Taylor [76] discusses an approach to the synthesis of sensor-based AL programs from task-level specifications. Taylor's method relies on representing prototypical motion strategies for particular tasks as parameterized robot programs, known as *procedure skeletons*. A skeleton has all the motions, error tests, and computations needed to carry out a task, but many of the parameters needed to specify motions and tests remain to be specified. The applicability of a particular skeleton to a task depends on the presence of certain features in the model and the values of parameters such as clearances and uncertainties. Choices among alternative strategies for a single operation are made by first computing the values of a set of parameters specific to the task, such as the magnitude of uncertainty region for the peg in peg-in-hole insertion, and then using these parameters to choose the best, that is fastest, strategy. Having chosen a strategy, the planner computes the additional parameters needed to specify the strategy motions, such as grasp positions and approach positions. A program is produced by inserting these parameters into the procedure skeleton that implements the chosen strategy.

Taylor's work on making planning decisions by manipulating constraints on positions that explicitly model error was significantly extended by Brooks [82]. The principal extension was the use of these symbolic constraints not only forward to get error bounds but backward to restrict the choices on plan variables and to introduce appropriate sensing into the program. This approach underlies much of TWAIN and is described in detail in Section 2.

The approach to strategy synthesis based on procedure skeletons assumes that task geometry for common sub-tasks is predictable and can be divided into a manageable number of classes each requiring a different skeleton. This assumption is needed because the sequence of motions in the skeleton will be consistent

only with a particular class of geometries. The assumption does not seem to be true in general. In particular, the presence of additional surfaces in tasks may generate unexpected contacts, leading to failures. This approach is in contrast to an approach which derives the strategy directly from consideration of the task description [Lozano-Pérez, Mason, and Taylor 83]. In the TWAIN design, both types of approaches play a role.

The LAMA system was designed at MIT [Lozano-Pérez 76, Lozano-Pérez and Winston 77] as a task-level language, but only partially implemented. LAMA formulated the relationship of task specification, obstacle avoidance, grasping, skeleton-based strategy synthesis, and error detection within one system. More recent work at MIT has explored issues in task planning in more detail outside of the context of any particular system [Brooks 82, 83a, 83b, Brooks and Lozano-Pérez 83, Lozano-Pérez 81, 83, Lozano-Pérez, Mason, and Taylor 84, Mason 81, 82].

AUTOPASS, at IBM [Lieberman and Wesley 77], defined the syntax and semantics of a task-level language and an approach to its implementation. A subset of the most general operation, the PLACE statement, was implemented. The major part of the implementation effort focused on a method for planning collision-free paths for Cartesian robots among polyhedral obstacles [Lozano-Pérez and Wesley 79].

RAPT [Popplestone, Ambler, and Bellos 78] is an implemented system for transforming symbolic specifications of geometric goals, together with a program which specifies the directions of the motions but not their length, into a sequence of end-effector positions. RAPT's emphasis has been primarily on task specification; it has not dealt yet with obstacle-avoidance, automatic grasping, or sensory operations.

Some robot-level language systems have proposed extensions to allow some task-level specifications. LM-GEO is an implemented extension to LM [Latombe and Mazer 81] which incorporates symbolic specifications of destinations. The specification of ROBEX [Weck and Zuhlke 81] includes the ability to plan automatically collision-free motions and to generate programs that use sensory information available during execution. A full-blown ROBEX, including these capabilities, has not been implemented.

2. Overview of the TWAIN System

The geometry of the world determines how TWAIN refines a given sequence of object motions into a detailed plan, including sensing steps, of action to be carried out by a manipulator.

Because of the complex interactions between plan steps, TWAIN must refine each step with as little commitment as possible to decisions within that refinement until the effects of decisions made in refining other plan steps are known. One approach might be to generate all the constraints that each plan step implies for other steps, and ultimately pick a set of motions and sense operations satisfying all those constraints. However, because of the number of possibilities in refining a single step and the inter-dependencies between steps, we cannot in general obtain

a complete set of constraints without making some decisions within individual plan steps. Hence the plan will fail if these decisions are incorrect. The planner therefore needs to employ a backtracking mechanism where decisions can be made and later undone in case of failure.

The TWAIN approach is first to model all plan steps at a broad level (as instantiated plan skeletons) and generate all constraints which the steps imply for every one of their possible refinements. The effects of the constraints are propagated throughout the complete plan. Each plan step is then refined into further detail. Again constraints are propagated and, in case of failure, dependency-directed backtracking occurs.

The order in which plan steps are refined depends on the type of operation they describe and their sensitivity to decisions made elsewhere in the plan. For example, gross motion planning is done last as it is almost completely insensitive to any fine-motion strategies, or sensing operations planned elsewhere. On rare occasions the choice of sensor location or jig layout for some force-directed motion strategy might block the workspace and make gross motion planning impossible. On the other hand the position of a workpiece relative to a sensor can affect whether the sensor can make useful measurements on it. Thus sensing operations should be planned before the workspace is layed out, so that constraints from those operations can be taken into account.

2.1. Constraints as a communication mechanism

To achieve our goals, it is neccessary to represent and manipulate geometric constraints and to distinguish between that which is not yet decided in the planning process and that which cannot be known even at plan execution time due to manipulator error, sensor error, and parts tolerances. TWAIN uses a scheme presented by [Brooks 82].

Constraints are represented by inequalities on explicit expressions over formal variables. These variables are of three types: *physical quantity* variables, *plan* variables, and *uncertainty variables.* There are two legal classes of expressions over these types: expressions which can include both plan and uncertainty variables, and expressions containing only physical quantities. The two classes of expressions can appear on opposite sides of an inequality or an equality. The semantics of the variables are as follows.

Physical quantity variables represent actual physical quantities in the real world. They are used to label quantities in the geometric world model.

Plan variables represent quantities whose values must be decided by the time of plan execution. Values for them are chosen as part of the planning process. Plan variables provide a mechanism for deferring decisions during planning, and constraints including plan variables provide a representation for reasoning about the implications of how those decisions will turn out. Often plan variables are used to represent nominal values for physical quantities.

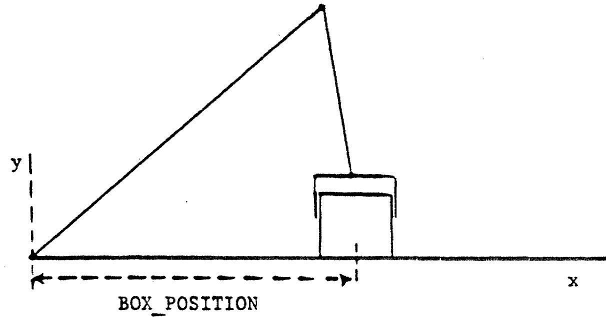

Figure 3. Definition of BOX-POSITION

Uncertainty variables represent quantities whose values can never be known, even at plan execution time; usually they represent the difference between a physical quantity and the nominal value which will be chosen for it by the planner. Thus they represent uncertainties in the planner's, and plan executor's, knowledge of the state of the world. Although their values can never be known, bounds on their magnitudes can either be known a priori (e.g. the manufacturing tolerance of a workpiece) or can be deduced by reasoning about the motions and sense operations leading up to the establishment of the appropriate physical quantity.

To illustrate these concepts consider the two-dimensional situation shown in figure 3. A workpiece is to be placed somewhere on a table by a two-link manipulator. Suppose the error, e in the x direction made by the manipulator in trying to place the box with a planned x coordinate p_x is given by

$$-0.2 + 0.005 \times p_x \leq e \leq 0.2 - 0.005 \times p_x. \tag{1}$$

Further, suppose that the working range of the manipulator is such that commanded positions for the box must satisfy

$$5 \leq p_x \leq 20. \tag{2}$$

We represent the constraints so placed on the physical situation by using three variables:

BOX-POSITION: a physical quantity variable representing the actual physical value which will be achieved as the x coordinate of where the box will be placed.

BOX-NOM: a plan variable representing the nominal value which will be chosen during planning and will be passed to the robot controller as the destination position commanded for the manipulator's motion, while grasping the box.

BOX-UNC: an uncertainty variable representing the error which will be made by the manipulator in placement of the box.

The above definitions imply

$$\text{BOX-POSITION} = \text{BOX-NOM} + \text{BOX-UNC}. \tag{3}$$

This provides a link from the geometric representation of the world, where BOX-POSITION is a parameter, to the constraint language which will be used in reasoning during the planning process.

The facts about the manipulator mentioned above when applied to the physical action of moving the box to destination give two constraints on the plan and uncertainty variable. The reach of the manipulator determines plausible planned locations for the box, i.e.

$$5 \leq \text{BOX-NOM} \leq 20$$

and the manipulator accuracy determines bounds on the uncertainty in the physical location of the box as a function of the commanded position for it, i.e.

$$-0.2 + 0.005 \times \text{BOX-NOM} \leq \text{BOX-UNC} \leq 0.2 - 0.005 \times \text{BOX-NOM}. \quad (4)$$

Given a suitable inference engine, such as described in [Brooks 81], we can now make deductions useful for the planning process. For instance, no matter which legal value is chosen for the nominal box position, the uncertainty about where it will actually be placed is no bigger than ± 0.175. Conversely, if during the planning process it becomes necessary to ensure that the position of the box is known to ± 0.15 then two possibilities are that the nominal position be constrained by

$$10 \leq \text{BOX-NOM}$$

or by introducing a sense operation immediately after placement of the box to see where it was placed. In the latter case the sensor must be chosen, and the nominal value of the box position may need to be constrained, so that the box will be within a region where the sensor error is bounded by ± 0.15.

Geometric models, sets of equalities such as (3) and inequalities such as (4) combine to form representations of classes of physical situations. Each physical situation in a class corresponds to one or more points in the satisfying set of all the inequalities. Such points with different values for plan variables correspond to different refinements of a plan. Points with identical values for all plan variables, but different values for uncertainty variables correspond to different physical realizations of a single planned situation.

2.2. The modules

The task planner consists of a *constraint propagator* and a *skeleton matcher* along with three planning modules (other modules might be added). The planning modules are a *fine-motion planner*, a *grasp planner*, and a *gross-motion planner*. The planner also has a library of *plan skeletons*.

Constraints are propagated between plan steps, providing the propagator with constraints on layout and enabling it to introduce sensing steps when necessary. Initially plan steps are modeled by instantiated plan skeletons and later refined into more detail by the planning modules. Constraint propagation continues as the plan is refined, propagating the inter–relating constraints back and forth between plan steps.

Broadly speaking, each planning module is given a class of situations for which it must produce a detailed plan step. The planning module can:

- Fail with no reason given.

- Fail giving a reason and perhaps incorporating a suggestion to change the given situation.
- Succeed, producing a detailed plan and perhaps a further set of constraints which should be applied to the given situation in order to refine it further.

Furthermore, each planning module can act as a generator of success/restriction pairs. As it is re–invoked it generates another solution or, eventually, fails. The generators define the search space for the constraint-propagation-triggered backtracking. The restrictions associated with each solution give the backtracking dependency direction [Stallman and Sussman 77].

2.3. Control structure

The task-level plan specification consists of a series of changes in location for objects in the workspace. These can be deduced by comparing consecutive world descriptions. The task planner then proceeds with the following phases:

1. The executive turns each motion into a sequence consisting of

   a. Gross manipulator motion

   b. Grasp

   c. Gross manipulator motion

   d. Fine motion

   e. Ungrasp.

2. Every one of these plan steps is then instantiated by a skeleton. This is done by the skeleton matcher. It is guaranteed to find a skeleton which matches. All gross manipulator motions are matched by the same skeleton as are grasps and ungrasps. Some fine motions will be matched by specific skeletons (such as "bolt in hole") and those that are not, are matched by a catch–all skeleton for *synthesized fine motion* (SFM). Skeleton matching includes identification of different physical locations. Each location is associated with the plan step where it first gains physical meaning (e.g., the position of an object on a table first becomes real during the put-down fine motion of the manipulator).

3. A dependency analysis is carried out for locations, that is, it is determined which physical placements of objects depend on others. The analysis forms multiple fibers running through a linear arrangement of plan steps — sometimes skipping over one or more steps.

4. Pre–condition and propagation constraints are propagated through the plan from one plan step to another. Sometimes the pre–conditions (called *applicability constraints*) will not be guaranteed to be satisfied. In that case backing up may be necessary. The dependency graph is followed. Backing up can introduce sensing. Skeletons for synthesized fine motion cannot be backed through directly. It is possible, however, to make certain inferences concerning whether sensing will be necessary and minimum bounds on costs of sensing. These are used to direct the backtracking. Note that this phase does not guarantee a workable plan, even at this

level of abstraction. It is essentially a branch-and-bound search and later it may turn out that some bounds were incorrect and more sensing will be necessary in non-SFM steps than is produced at this point. If a fine-motion skeleton (non-SFM) causes failure, then the skeleton generator for that motion is re-invoked, a new skeleton is selected, and this phase is repeated.

5. We now have some constraints on many of the locations. With these we do a coarse layout, identifying large reasonable areas to place planned locations. For each SFM step the manipulator error is analyzed over these large areas, which may be partitioned into subsets having widely different error ball upper bounds. The partitions are ordered smallest error bound first, and a list of pairs of error balls and work areas is formed. These will be used to drive generators for syntheses of fine-motion strategies. If all possible gross layouts have already been tried then the planner fails.

6. For each SFM generator look at its first error ball and work area, but do not remove them from the generator list. If an actual fine-motion synthesis (see section 3) has not yet been done for this plan step then do it. The result is a series of sets R from which the synthesized fine motion is guaranteed to succeed (see section 3). The fine-motion planner also generates a guard volume where the motion will happen. This volume is the subject of the layout phase (phase 10). If an SFM generator is already exhausted then backup to phase 5 for a new gross layout.

7. Using branch-and-bound search on cost estimates propagate constraints through the plan with the old SFM skeletons replaced by actual fine motions. Choose the smallest set R (of initial locations) for each fine motion that can be guaranteed to be reached (i.e. the manipulator error ball fits inside it). If some fine motion causes plan failure then remove it from its generator and backup to phase 6 for a new synthesis.

8. For each object that must be grasped generate a list of possible grasp configurations, ordered by criteria such as firmness of grip and accessibility of the grasp surfaces. A grasp depends on both the initial configuration of the object and its goal configuration. The interaction of the hand with the fine motion at the end of any gross motion of the object must also be considered. On failure to find any grasp at all some new terminal strategy must be necessary, so backup to phase 6 for a new fine-motion synthesis. The actual positions of the grasp might depend on plan variables. Each grasp produces a guard volume which must be free of obstacles so that the hand can fit in a position necessary to achieve the grasp. Note that the guard volume, appropriately translated and perhaps re-oriented, must be free both at pick up and at put down. The first grasp in the list of grasps is used in subsequent planning phases, although they may back up to this phase for a new grasp.

9. Propagate constraints through the plan moving forward until some pre-condition constraints are not met, and then backing up to introduce sensing or otherwise constrain plan variables. This phase is essentially a repeat of phase 4, except that we have more detailed instantiations of each plan step. Thus we can

now back through fine-motion steps, and perhaps introduce sensing steps to achieve the pre–conditions necessary for the success of such steps. Failure during constraint propagation can only be due to a synthesized fine motion having excessively strong pre–conditions. In that case planning is backed up to phase 6 for a new fine motion synthesis.

10. Choose actual physical locations for the initial location of every object and for the nominal locations to be used in all intermediate steps of the plan. This is the detailed layout of the workspace. On failure due to clutter of some work area by numerous objects and tasks (such as may happen in an area where the manipulator is most accurate) backup to phase 5 for a new gross layout. The result will be that new, more difficult, fine motion syntheses will have to be carried out which will succeed in areas with less accurate manipulator characteristics. On failure due to the impossibility of keeping a guard volume free for the hand return to phase 8 for the next available grasp.

11. Plan collision-free motions for the manipulator to achieve each gross change in location of objects, and to move the manipulator from the release of one object to the acquisition of the next. If gross-motion planning fails then backup to one of phases 5 (gross layout), 8 (grasp analysis) or 10 (detailed layout), depending on the reason for failure. If no path could be found for the payload due to re–orientation difficulties then variously try tweaking detailed layout in phase 10 (such as making wider corridors if free space at the failed path segments) or retry the grasp analysis at phase 8 (such as selecting candidate grasps which place a widely different part of the object along center of rotation of the last manipulator joint axis). If collisions with the upperarm or forearm were the main problem then backup to phase 5 for a new gross layout of the workspace.

2.4. Skeletons

A plan skeleton models a step in a plan in terms of its inputs and outputs. The idea is that the details of the plan step can be treated as a black box by the rest of the plan — the rest being affected only by the input/output behavior of an individual plan step.

Skeletons can describe plan steps at different levels of detail. For instance there might be a skeleton for pickup used early in the planning process. Later in the process the action might be modeled by a series of four finer plan steps each instantiated by a skeleton, gross-manipulator motion, fine-manipulator motion, grasp, and another gross-manipulator motion.

A skeleton is specified by a geometric description of objects and their state in a world and by two sets of constraints: a set of *applicability constraints* and a set of *propagation constraints*. The skeleton is instantiated by finding a match between the geometric description and the known world state. Both sets of constraints in the skeleton are expressed in variables which get instantiated by physical quantity variables as a by-product of the geometric matching.

Once instantiated, the feasibility of using such a plan step can be considered. If the applicability constraints are satisfied by a physical situation, then the plan

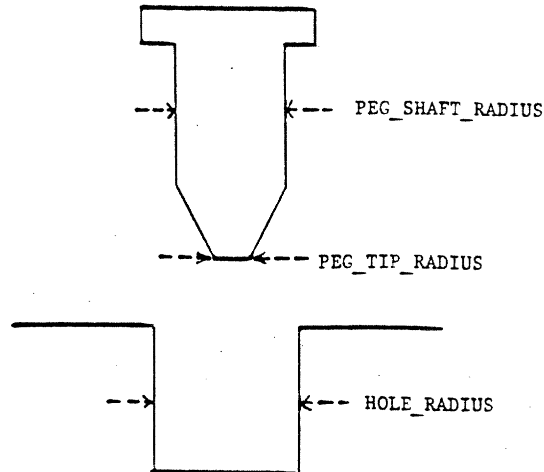

Figure 4. Bolt-in-hole example.

step modeled by the skeleton is guaranteed to succeed. The propagation constraints put bounds on the possible physical situations that can be produced by the plan step given any initial situation satisfying the applicability constraints.

Consider a skeleton to describe insertion of a peg into a hole. For simplicity we will assume a two-dimensional world, ignoring the details of vertical motions even in that world. To match a physical situation and plan step there must be a bolt and a hole. The applicability constraint set consists of

$$\text{PEG-SHAFT-RADIUS} \leq \text{HOLE-RADIUS},$$

$$\begin{aligned} &\text{PEG-TIP-RADIUS} - \text{HOLE-RADIUS} \\ \leq &\text{HOLE-POSITION} - \text{PEG-POSITION} \\ \leq &\text{HOLE-RADIUS} - \text{PEG-TIP-RADIUS}, \end{aligned}$$

and

$$\textit{nominal}(\text{HOLE-POSITION}) = \textit{nominal}(\text{PEG-POSITION}).$$

The propagation constraint set is simply:

$$\begin{aligned} &\text{PEG-SHAFT-RADIUS} - \text{HOLE-RADIUS} \\ \leq &\text{HOLE-POSITION} - \text{PEG-RESULT-POSITION} \\ \leq &\text{HOLE-RADIUS} - \text{PEG-SHAFT-RADIUS} \end{aligned}$$

and

$$\textit{nominal}(\text{HOLE-POSITION}) = \textit{nominal}(\text{PEG-RESULT-POSITION}).$$

The meta–function *nominal* refers to the planned value for a physical quantity.

Now consider the physical situation shown in figure 4. The peg is labeled A and the hole is labeled B. In this example, both the peg and hole have definite radii. After matching the skeleton to the physical situation, the constraint sets are instantiated as; the applicability constraints:

$$0.2 \leq 0.25,$$

which is trivially true,

$$-0.20 \leq \text{B-POSITION} - \text{A-POSITION} \leq 0.2$$

and

$$nominal(\text{B-POSITION}) = nominal(\text{A-POSITION}),$$

while the propagation constraints are:

$$-0.05 \leq \text{B-POSITION} - \text{A-RESULT-POSITION} \leq 0.05$$

and

$$nominal(\text{B-POSITION}) = nominal(\text{A-RESULT-POSITION}).$$

The physical quantities are expanded into plan and uncertainty components using

$$\text{A-POSITION} = \text{B-NOM} + \text{A-UNC}$$
$$\text{B-POSITION} = \text{B-NOM} + \text{B-UNC}$$
$$\text{A-RESULT-POSITION} = \text{B-NOM} + \text{A-RESULT-UNC}$$

where the equalities in the constraints are used to substitute equal values. The choice of using B-NOM as the nominal value for all three physical quantities is for readability only — internally it is more likely to be something like G0487. The constraint equalities have dictated that the peg should be nominally aligned with the hole before the insertion. This is manifested by the peg and hole being given the same nominal value in this plan step. The constraint propagator forces this to be true by, for instance, specifying the nominal end point of the previous gross motion that moves peg to the vicinity of the hole.

Now the applicability constraints reduce to:

$$-0.20 \leq \text{B-UNC} - \text{A-UNC} \leq 0.2$$

and the propagation constraints to:

$$-0.05 \leq \text{B-UNC} - \text{A-RESULT-UNC} \leq 0.05.$$

The methods developed in [Brooks 82] can be used to find ways of guaranteeing that the applicability constraint are met. For instance it may be necessary to sense the position of the hole before a previous plan step of moving the peg to a nominal position above the hole. The sense step would give a new more accurate estimate of the location of the hole giving a new B-NOM value and a better bound on the possible values for B-UNC.

Given a range of values for B-UNC the propagation constraint determines a range of values for A-RESULT-UNC, the uncertainty in peg position after insertion. Later steps in the plan may need to examine this range to see whether it meets their applicability constraints.

Skeletons for many other actions can be produced similarly. They need not be used only to model actions in detail. The constraint sets might be very underconstraining allowing minimal requirements for a large class of actions to be simultaneously considered. Such skeletons must later be replaced by detailed plans produced by one of the specialist planning modules.

A plausible small library of skeletons includes:

1. Gross Motion
2. Grasp
3. Bolt in hole
4. Ungrasp
5. Dead reckon vertical align (i.e. stacking)
6. Catch–all *synthesized fine motion*

2.5. The planner modules in more detail

The three planning modules are briefly described here in general form. In sections 3 through 5 of the proposal we refer to published work which describes implementations of specializations of each of the module descriptions.

All modules have access to a current context world model, which has specifications via geometric descriptions and algebraic constraints of all the parts whose locations have been decided exactly or at least been constrained to some set of possibilities.

1. *Fine Motion*
Input:

a. Initial locations of parts

b. Goal specification

c. Assembly constraints

d. Error bounds;

Output:

a. Compliant motion strategy

b. Legal initial locations for each motion

c. Guard volume.

The fine-motion planner is given as inputs the sets of possible initial locations for the parts and a specification of the range of legal final configurations. It is also given bounds on the errors in measuring the manipulator's position and the forces acting on it, and bounds on the error in controlling the position and velocity of the manipulator. Lastly, it may be given constraints on the motions, such constraints might forbid hitting some surfaces or exceeding bounds on forces.

The planner determines compliant motions which ensure that the parts will reach one of the final configurations if started from within the specified range of initial locations. Typically the planning process will also place additional constraints on the legal initial locations.

Figure 7 shows an example of fine-motion planning for a point in two dimensions. This is actually the configuration space of a simple block-in-corner task (see the Appendix). The task is to move the point p from its initial location to somewhere in set G. A single compliant motion along the direction v from anywhere within the shaded subset of A will reach some part of G. Note that due to manipulator

error we can only specify the position of the point to be within some error ball and the specified heading within some cone around v. The planner must pick the initial position and the direction of motion with these errors in mind.

In general, achieving a task will require several motions. For each of these a region of initial locations (and corresponding motions) will be computed. The planner returns each of these regions as well as the corresponding motions. These regions can serve as alternative goals for the gross motion planner.

The fine-motion planner assumes it has a complete description of the environment. In TWAIN, however, fine-motion planning is done before a final layout of the environment is available. The planner computes a *guard volume* where the introduction of another object would affect the planned strategy. These guard volumes are used in the final layout phase.

2. *Grasping*
Input:

a. Pick up location of part

b. Put down destination of part

c. Grasp constraints

d. Error bounds;

Output:

a. Grasp configurations

b. Grasp motions

c. Guard volume.

The grasp planner is given as inputs the pick up and put down locations for the part to be grasped (possibly in terms of plan variables). It is also given bounds on the position error of the part at the pick up location. As in fine motion planning, the planner may be given constraints on where the part should, or should not be grasped.

The planner determines where the part should be grasped and the desired orientation of the hand relative to the surface. The planner also must determine a sequence of motions that guarantees that the desired grasp configuration is reached in the presence of errors in manipulator positioning and part location.

Figure 8 shows a simple example of grasp planning for a parallel jaw gripper. Note that the choice of where to place the fingers depends not only on the part, but also on the environment near the part at the initial and final locations. As with fine motion planning, the grasp planner specifies a guard volume in which the presence of objects would require pre-planning.

3. *Gross Motion*
Input:

a. Start locations

b. Goal locations

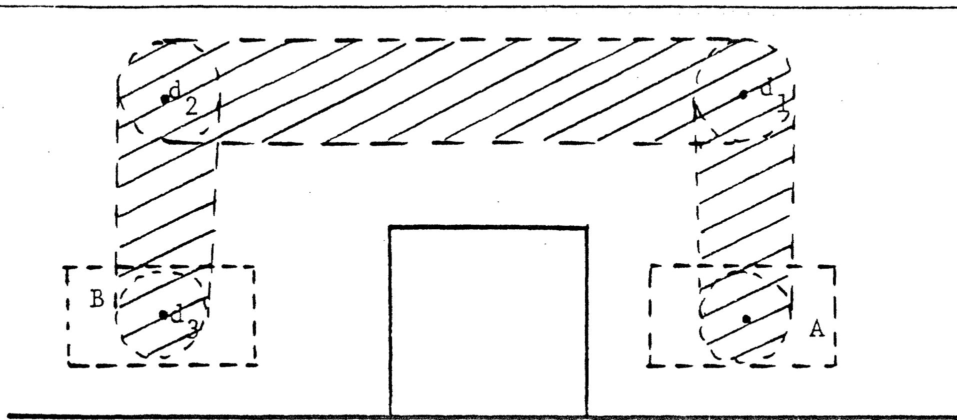

Figure 5. Gross-motion planning.

c. Trajectory constraints

d. Error bounds;

Output:

a. Path (may be null)

b. Restricted start locations

c. Restricted goal locations.

The gross-motion planner determines collision-free motions for the manipulator as it moves to the general location of an object in order to grasp it and as it transfers an object within the workspace. The two classes of motions can be treated by a single algorithm — the volume occupied by the hand changes depending on what is grasped.

The planner is given initial and goal sets of configurations for the hand. It is given bounds (perhaps parameterized) on the location error of any payload relative to the hand, and on the position control error of the manipulator. Lastly it may be given constraints which must be met by the trajectory. These are of two types. They can limit the class of trajectories considered (e.g. it might be demanded that the payload be reoriented only about a vertical axis), or they can provide a criterion for choosing the trajectory from the considered class (e.g. minimization of trajectory length, or of payload reorientation).

The planner produces a set of gross-manipulator motions (perhaps parameterized in terms of plan variables) which guarantee motion of the hand or payload from any point in the initial set of configurations to some point in the final set.

Figure 5 shows an example of gross-motion control for moving a point in two dimensions (this might be the configuration space of some moving object with a more interesting shape). The task is to move the point from any position in set A to somewhere in set B. Three commands are generated: "move to d_1", "move to d_2" and "move to d_3". Each is a command to lower level servo routines to move in a straight line to the desired point (using pure position control). Due to manipulator inaccuracies the true destinations of the motion commands will lie in the error balls D_1, D_2 and D_3 respectively.

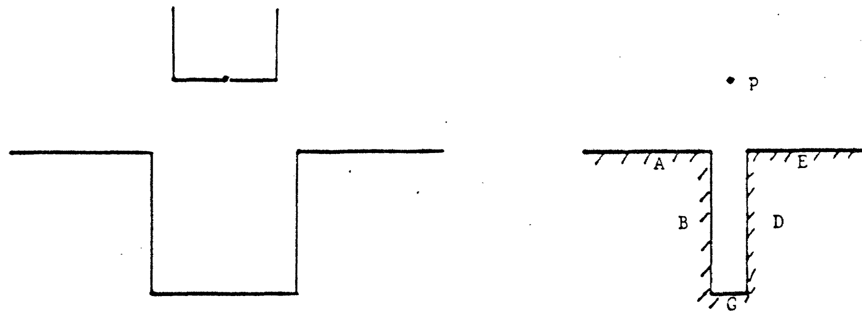

Figure 6. Peg-in-hole: (a) original formulation (b) transformed to point problem.

Notice that D_3 is wholly contained in set B, so if it is reached the gross motion will have been successful.

Due to manipulator inaccuracies, straight-line motions will not necessarily be achieved. The shaded regions, in figure 5, show all possible paths, demonstrating that whatever motion actually occurs is guaranteed to be collision free.

3. Fine-Motion Synthesis

The problem of automatically synthesizing a fine motion from a geometric description has received little attention in the literature. Previous approaches were either based purely on skeletons [Taylor 76, Lozano-Pérez 76] or on learning strategies [Dufay and Latombe 83]. In this section we outline an approach to automatic fine-motion synthesis initially described in [Lozano-Pérez, Mason, and Taylor 84] and further developed in [Mason 84, Erdmann 84]. This approach is based on the notion of a *configuration space* [Lozano-Pérez 81, 83, Brooks and Lozano-Pérez 83]; see the Appendix for a brief introduction.

The fine-motion planner constructs a sequence of motions that guarantees that some configuration from a specified range of goal configurations will be reached from anywhere within a computed range of start configurations. Each element in the motion sequence is a *guarded compliant motion*, in particular, a commanded velocity vector for a *generalized damper* [Whitney 76], and a termination predicate. The desired motion for a generalized damper is determined by the following relationship

$$\mathrm{f} = \mathrm{B}(\mathrm{v} - \mathrm{v}_0)$$

where f is the vector of forces acting on the moving object, v_0 is the commanded velocity vector, and v is the actual velocity vector. The effect of these compliant motions is to slide on the C-surfaces (see the Appendix) derived from the obstacles. When not in contact with a surface, the motion will be along the commanded velocity (within some velocity uncertainty). The motion terminates when the associated predicate (a function of observed configuration, velocity, and elapsed time) evaluates to true.

Consider the simple task of moving the point p from its initial configuration to any one of the configurations in G (see Figure 6(b)). This is the C-space version

of the two-dimensional peg-in-hole problem in Figure 6(a) when the axes of the peg and hole are constrained to be parallel. The basic step in the synthesis approach is to identify ranges of configurations from where p can reach G by a single motion. The directions of such motions can be represented as unit vectors, $\mathbf{v}_i$. For each $\mathbf{v}_i$, we can compute all those configurations, P_i, such that a motion along $\mathbf{v}_i$ from that configuration would reach some point of G. We call this range of configurations that can reach the goal by a single motion along a specified heading vector the *pre-image* of the goal (for that vector). All we need do to guarantee that p reaches G from any point in any of the P_i is to use $\mathbf{v}_i$ as the commanded velocity vector for a damper.

The computation of pre-images must take into account the possibility of the moving object sticking on a surface. In particular, assuming the motion is generated by a damper (with $\mathbf{B} = b\mathbf{I}$), if the angle between the commanded velocity and the normal of a surface is less than the *friction angle* (the arctangent of the coefficient of friction) then the motion will stick on that surface.

If no pre-image of G contains the peg's current configuration, then we can apply the same pre-image computation recursively using each of the existing pre-images as a possible goal. Each pre-image of G, P_i, serves to define a new goal set. This process is repeated until some pre-image contains some subset of the legal start configurations. From the chain of pre-images we can construct a motion sequence.

One key problem in the synthesis method is to discover the sequence of command velocity vectors $\mathbf{v}_i$. Our approach is to narrow in on feasible values of $\mathbf{v}_i$ by progressive refinement. We start with the complete range of possible $\mathbf{v}_i$'s and remove from that range any values that can possibly lead to failure (by sticking on a non-goal surface or by sliding away from the goal). At each step of the algorithm, we compute the pre-image of the goal for the current *range* of $\mathbf{v}_i$'s. The pre-image for a range of commanded velocities is the intersection of the pre-images for each of the velocities. These are the configurations guaranteed to reach the goal for *all* the velocities in the range. If the pre-image includes feasible starting configurations, then we have found a valid motion sequence, otherwise the current range of velocities must be narrowed further.

Figure 7 illustrates the method on a simple two-dimensional block-in-corner example. The directed graph shown there has nodes for each of the C-surfaces in the task and one node representing free C-space (C). There is a link from nodes m to n in the graph if some velocity in the current range may cause the robot to move from some point in m (which is not in n) to some point in n (which may be at the intersection of m and n) without going through points in any other node. We call this the *reachability graph* for that range of commanded velocities.

In the example, we start out with a range of commanded velocities including any velocity that will move p from nearby points onto the goal G (we diagram ranges of commanded velocities as sectors of a circle). The reachability graph for this range of velocities is shown at the top of Figure 7. In this figure, we have indicated those surfaces where the moving object may stick (using the electrical ground symbol). The (potentially) sticking surfaces are those whose friction cones

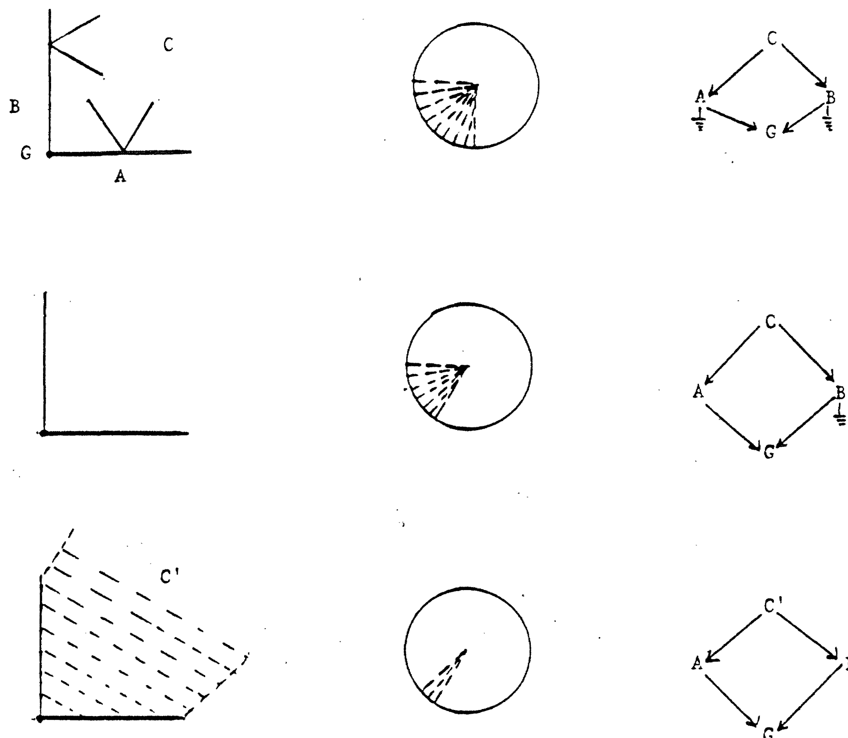

Figure 7. Block-in-corner fine-motion synthesis.

overlap the current velocity range. No pre-image exists for the initial velocity range because of the possibility of sticking on either A or B. By removing velocities that may cause sticking on surface A or B from the current range of velocities, we obtain a velocity range for which a pre-image (C') exists. This pre-image intersects the initial range of configurations, so a successful motion has been found.

The example above can be done with a single motion. We did not require recursive calls to the planner. In general, we have a choice of refining the range of directions or of using the current pre-image as the goal for a recursive call to the same algorithm. This choice at each step defines the search space of motion sequences. Another important aspect of the approach is the synthesis of termination predicates for the motions. These issues are further discussed in [Lozano-Pérez, Mason, and Taylor 84, Mason 84, Erdmann 84].

4. Grasping

The problem of choosing a grasp point on an object has received significant attention in the literature [Paul 72, Lozano-Pérez 76, 81 Wingham 77, Brou 80, Laugier 81, Mason 82, Laugier and Pertin 83]. The approach to grasping described here is based on that described in [Lozano-Pérez 81]; a more detailed treatment can be found there. This approach is also based on the notion of configuration space; see Appendix.

The grasp planner chooses which object surfaces will be grasped and builds a description of the grasp configurations (on those surfaces) that satisfy the following constraints:

1. The inside of the fingers are in contact with surfaces of P, the object to be grasped.
2. There are no collisions between the manipulator hand and any nearby objects for any possible start configuration of P.
3. There are no collisions between the manipulator hand and any nearby objects for any possible goal configuration of P.
4. The grasp is stable, i.e., can withstand forces generated during motion and assembly.

We assume that the manipulator hand is a parallel jaw. We further assume that the manipulator can be partitioned into an arm and hand. The arm serves to place the wrist at any point in the workspace; the hand determines the final configuration of the gripper. This is a common kinematics for manipulators and has a number of theoretical and practical advantages. The grasp planner determines candidate hand configurations; the gross-motion planner must then pick some hand configuration that allows finding a collision-free path from the start to the goal.

The choice of grasp surfaces is done by ranking the surfaces by their likelihood of providing a stable grasp and then choosing the highest ranked surface that leads to a feasible grasp configuration. A general treatment of stability in grasping is not yet available, although some promising approaches exist [Hanafusa and Asada 77]. When the object to be grasped is small relative to the manipulator hand, two simple heuristics provide a fair chance of identifying a stable grasp (see also [Paul 72, Brou 80]). The heuristics are:

1. Ensure at least a minimum contact area of the fingers with the grasp surfaces. The amount of overlap should depend on object properties such as weight and surface smoothness.
2. The perpendicular projection of P's center of mass should be near to the contact area of the fingers and grasp surfaces.

The grasp planner computes feasible grasp configurations for the top ranked candidate grasp surfaces. Note that because the manipulator configuration will change while moving P from its start to its goal configuration, we represent the grasp configurations as the configuration of the hand relative to P. We can impose restrictions that reduce the dimensionality of the set of grasp configurations. One simple restriction, for parallel jaw hands, is to require that at least one of the surfaces grasped be planar (the other may be a planar surface, curved surface, edge, or vertex).

Let P_i be the planar face of P to be grasped, P_j is the other face (edge or vertex), and F_1 and F_2 be the inside faces of the manipulator's fingers. Under the restriction stated above, when P is grasped, either F_1 or F_2 is coplanar with P_i (without loss of generality assume F_1 is coplanar with P_i). Under these conditions,

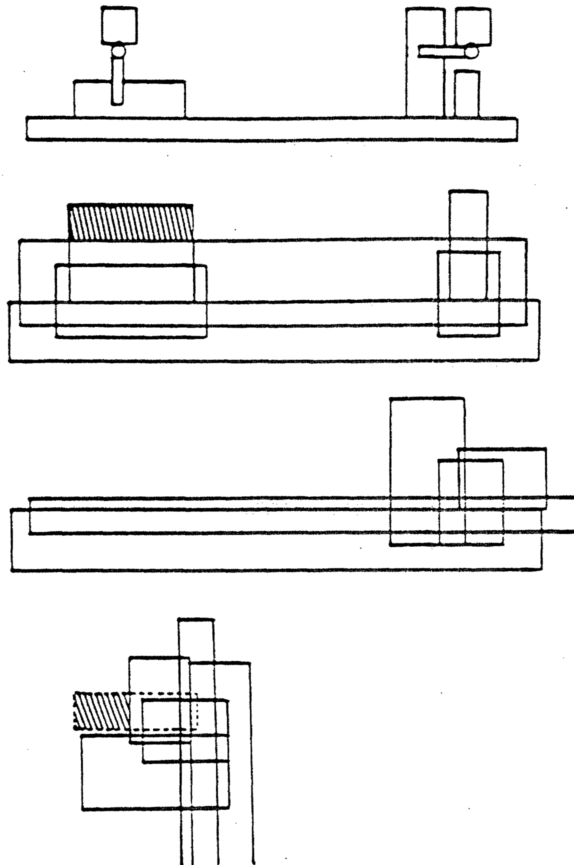

Figure 8. Grasp set computations.

the legal (x, y, z) positions of all points on the hand are restricted to be on some plane parallel to P_i. The hand may rotate about the normal to this plane. Let G be the set of configurations of the hand for which P_i and F_1 are coplanar. G is called the *grasp set* for P_i.

Not all the configurations in G are feasible grasp configurations, either because the fingers are not in contact with the grasp surfaces or because the corresponding manipulator configuration causes a collision (at the start or at the goal). The constraint that the fingers touch the grasp surfaces can be readily enforced by restricting the grasp set to be the intersection of those hand configurations for which F_1 overlaps P_i and those for which F_2 overlaps P_j. This intersection set can be computed explicitly in low-dimensional C-space [Lozano-Pérez 81, 83, Brooks and Lozano-Perez 83]. Similarly, those hand configurations (defined relative to P) that cause collisions with objects at the start or at the goal can be computed. The grasp set G can then be intersected with their complement to obtain the set of feasible grasp configurations.

Figure 8 shows an example of the feasible grasp computation: (a) the pick-up and put-down orientations of the hand, (b) C-space obstacles at pick up (the shaded region is the accesible part of the grasp set), (c) C-space obstacles at put down, (d) put down obstacles constraining the grasp set further.

If P is free to move during the grasping operation and its initial position is not known to high accuracy, then the grasp planner must take into consideration the possible motions during planning. This is quite difficult in general; see [Mason 82].

Many grippers currently under development, such as Salisbury's three-fingered hand and the Utah-MIT hand, are much more complex than the parallel jaw gripper we have been considering. The planning method above, although still relevant, becomes vastly more complex for these multi-fingered hands. This is an area where more research is desperately needed.

5. Gross Motion

After a trickle of early work on collision-free gross-motion planning, there has been a avalanche of new ideas and developments recently.

The earliest reasonably general algorithms for manipulators were for the Stanford arm (it has one sliding and five revolute joints). One was implemented [Udupa 77] and the other partially implemented [Widdoes 74]. Both relied on approximations for the payload, limited wrist action, and tesselation of joint space to describe forbidden and free regions of real space. The problem with tesselation schemes is that to get adequate motion control a multi-dimensional space must be finely tesselated.

Lozano-Pérez [81] presented an implemented algorithm for Cartesian manipulators. Cartesian manipulators have three sliding joints whose axes are orthogonal and thus they can be used as the axes of space representation. Lozano-Pérez's algorithm used configuration space for the Cartesian portion of the manipulator (where the natural Euclidean axes of the configuration space correspond exactly to the joints of the manipulator) and subdivided ranges of angles for the hand, within each of which a bounding volume for the hand and payload is used.

Schwartz and Sharir [82] have shown that the problem is polynomial in the number of obstacle surfaces for a manipulator with a fixed number of joints. If n is the number of obstacle surfaces, the running time of their algorithm is $O(n^{64})$ for a six degree-of-freedom manipulator. It is, therefore, primarily of theoretical interest and not meant to be implemented.

Brooks and Lozano-Pérez [83] have developed a practical and implemented algorithm for polygonal obstacle avoidance which produces paths which require arbitrarily difficult rotations up to a preset resolution.

Brooks [83a] developed a new representation for two-dimensional free space as overlapping freeways. This has led [Brooks 83b] to the development and implementation of a gross motion algorithm for pick and place operations for a manipulator with revolute joints.

The key idea is that free space should be explicitly represented in such terms that it is easy to determine the collision-free motion segments that can be made

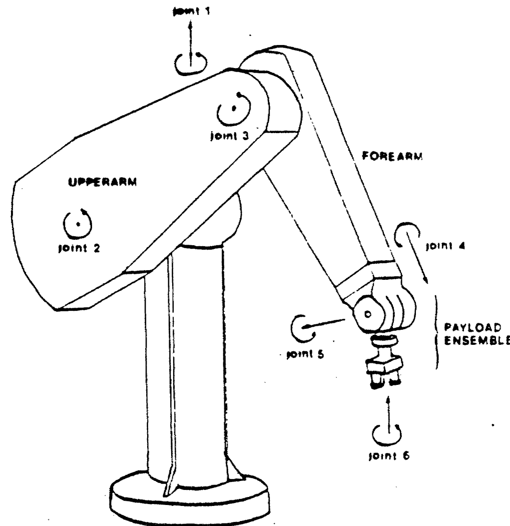

Figure 9. Unimation PUMA manipulator.

by the manipulator and its payload. Individual legal motion segments are linked to form a complete motion for the manipulator.

See figure 9 for a diagram of the PUMA. Brooks' algorithm decomposes the problem by spending two degrees of freedom of the manipulator to partially decouple the payload and the upperarm of the manipulator. In the six degree-of-freedom PUMA it keeps joint 4 fixed (there is no joint 4 for the 5 degree-of-freedom PUMA) and uses joint 5 to compensate for the payload orientation for the motions of the upper and forearms. Joint 6 is free to re-orient the payload about its vertical axis, but such re-orientation does not require motion of either the upper or the forearm — it is completely decoupled. This is only two-dimensional rotation. There is still coupling of translations of the payload and the motion of the upper links of the arm. A major new contribution of the algorithm is that motion of the components can at first be analyzed separately and then, later, constraints are propagated between the solutions to account for the remaining coupling.

The algorithm finds paths where the payload is moved in straight lines, either horizontal or vertical, and is only re-oriented by rotations about the vertical axis of the world coordinate system. Thus only 4 degrees of freedom are considered for the PUMA.

The payload and the hand are merged geometrically, and the payload is considered to be a prism, with convex cross section. The payload can rotate about the vertical, as joint 6 rotates.

Obstacles in the work space are of two types: those supported from below and those hanging from above. Both are prisms with convex cross sections. Non-convex obstacles can be modeled by overlapping prisms. Prisms can be supported from below if they rest on the workspace table or on one another as long as they are fully supported. Thus no point in free space ever has a bottom supported obstacle

above it. Such obstacle descriptions have been extracted from depth measurements from a stereo pair of overhead cameras. The algorithm has generated collision free paths from that data.

Similar pre–defined obstacles may also hang from above intruding into the workspace of the upperarm and forearm. Obstacles are precluded from a cylinder surrounding the manipulator base.

The class of motions allowed suffice for many assembly operations, and, with as yet unknown algorithms for re–orienting the payload without major arm motion, the algorithm could provide gross motion planning for all but the most difficult realistic problems.

6. Conclusions

In this proposal we have outlined an architecture for a new task-level system, which we call TWAIN. Our goal has been to define a unified framework for existing and future research on task planning. We have summarized approaches to several of the key problems in task planning: fine motion synthesis, grasping, and gross motion planning. These areas are relatively mature. Some other areas such as automatic parts layout, feeding, and fixturing have received significantly less attention. We propose to construct a prototype implementation of TWAIN during the next two years where the focus will be on the interaction between the modules described here.

6. References

Binford, T. O. (1979). *The AL Language for Intelligent Robots,* IRIA Seminar on Languages and Methods of Programming Industrial Robots, Rocquencourt, France, June.

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

Brooks, R. A. (1982). *Symbolic Error Analysis and Robot Planning,* International Journal of Robotics Research, vol 1, no. 4, Dec., 29–68.

Brooks, R. A. (1983a). *Solving the Find-Path Problem by Good Representation of Free Space,* IEEE Trans. on Systems, Man and Cybernetics (SMC–13):190–197.

Brooks, R. A. (1983b). *Planning Collision Free Motions for Pick and Place Operations,* First International Symposium on Robotics Research, Bretton Woods, New Hampshire, August.

Brooks, R. A. and T. Lozano–Pérez (1983). *A Subdivision Algorithm In Configuration Space For Findpath With Rotation,* IJCAI–83, Karlsruhe, Germany.

Brou, P. (1980). *Implementation of High-Level Commands for Robots,* M. S. thesis, MIT Dept. of Electrical Engineering and Computer Science, December.

Dufay, B. and J. C. Latombe (1983). *An Approach to Automatic Robot Programming Based on Inductive Learning,* First International Symposium on Robotics Research, Bretton Woods, August.

Erdmann, M. A. (1984). *On Motion Planning with Uncertainty,* MIT Artificial Intelligence Laboratory, Technical Report 810.

Feldman, J., et al. (1971). *The Stanford Hand-Eye Project,* First IJCAI, London, England, September.

Finkel, R., Taylor, R., Bolles, R., Paul, R., and Feldman, J. (1974). *AL, A programming system for automation,* Stanford Artificial Intelligence Laboratory, AIM-177, November.

Grossman, D. D. and Taylor, R. H. (1978). *Interactive Generation of Object Models with a Manipulator,* IEEE Transactions on Systems, Man, and Cybernetics (SMC-8):667–679.

Hanafusa, H., and Asada, H. (1977). *A robotic hand with elastic fingers and its application to assembly process,* IFAC Symposium on Information and Control Problems in Manufacturing Technology, Tokyo.

Latombe, J. C. and Mazer, E. (1981). *LM: a High-Level Language for Controlling Assembly Robots,* Eleventh International Symposium on Industrial Robots,Tokyo, Japan, October.

Laugier, C. (1981). *A program for automatic grasping of objects with a robot arm,* Eleventh International Symposium on Industrial Robots, Tokyo, Japan, October.

Laugier, C. and J. Pertin (1983). *Automatic Grasping: A Case Study in Accessibility Analysis,* Laboratoire IMAG, Report 342, January.

Lieberman, L.I., and Wesley, M. A. (1977). *AUTOPASS: an automatic programming system for computer controlled mechanical assembly,* IBM Journal of Research Development (21):321-333.

Lozano-Pérez, T. (1976). *The design of a mechanical assembly system,* MIT Artificial Intelligence Laboratory, TR 397, December.

Lozano–Pérez, T. (1981). *Automatic Planning of Manipulator Transfer Movements,* IEEE Trans. on Systems, Man and Cybernetics (SMC–11):681–698.

Lozano–Pérez, T. (1983a). *Spatial Planning: A Configuration Space Approach,* IEEE Trans. on Computers (C–32):108–120.

Lozano-Pérez, T., and Wesley, M. A. (1979). *An algorithm for planning collision-free paths among polyhedral obstacles,* Communications of the ACM (22):560-570.

Lozano-Pérez, T., and Winston, P. H. (1977). *LAMA: a language for automatic mechanical assembly,* Fifth International Joint Conference on Artificial Intelligence, Cambridge, Mass., August.

Lozano-Pérez, T., Mason, M. T., and Taylor, R. H. (1984). *Automatic Synthesis of Fine-Motion Strategies for Robots,* Int. J. of Robotics Research, vol 3, no. 1.

Mason, M.T. (1981). *Compliance and force control for computer controlled manipulators,* IEEE Transactions on Systems, Man and Cybernetics (SMC-11):418-432.

Mason, M. T. (1982). *Manipulator Grasping and Pushing Operations,* MIT Artificial Intelligence Laboratory, Technical Report 690.

Mason, M. T. (1984). *Automatic Planning of Fine Motions: Correctness and Completeness,* 1984 IEEE International Conference on Robotics, Atlanta Ga..

Paul, R. P. (1972). *Modelling, trajectory calculation, and servoing of a controlled arm,* Stanford Artificial Intelligence Laboratory, AIM 177, November.

Popplestone, R.J., Ambler, A. P., and Bellos, I. (1978). *RAPT, A language for describing assemblies,* Industrial Robot (5):131-137.

Schwartz, J. T. and M. Sharir (1982). *On the Piano Movers Problem II: General Properties for Computing Topological Properties of Real Algebraic Manifolds,* Department of Computer Science, Courant Institute of Mathematical Sciences, NYU, Report 41, February.

Stallman, R. M. and G. J. Sussman (1977). *Forwrad Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis,* Artificial Intelligence (9):135–196.

Taylor, R. H. (1976). *The Synthesis of Manipulator Control Programs from Task-level Specifications,* Stanford Artificial Intelligence Laboratory, AIM-282, July.

Udupa, S. M. (1977). *Collision Detection and Avoidance in Computer Controlled Manipulators,* Proceedings of IJCAI–5, MIT, Cambridge, Ma., Aug. 1977, 737-748.

Weck, M. and Zuhlke, D. (1981). *Fundamentals for the Development of a High-Level Programming Language for Numerically Controlled Industrial Robots,* AUTOFACT West, Dearborn, Michigan.

Whitney, D.E. (1976). *Force feedback control of manipulator fine motions,* J. Dynamic Systems, Measurement, Control, 91–97, June.

Widdoes, L. C. (1974). *Obstacle avoidance.,* A heuristic collision avoider for the Stanford robot arm.Unpublished memo, Stanford Artificial Intelligence Laboratory.

Wingham, M. (1977). *Planning how to grasp objects in a cluttered environment,* M. Ph. thesis, Edinburgh University.

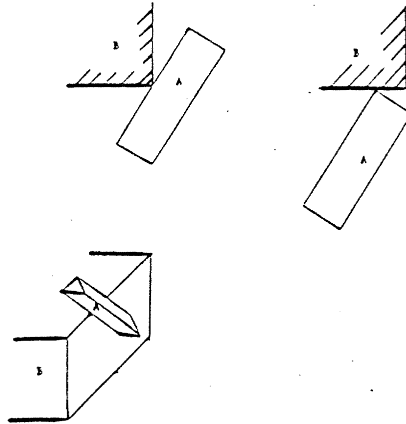

Figure 10. Geometric conditions giving rise to C-surfaces.

Appendix: Configuration Space

A *configuration* of an object is the set of parameters needed to specify completely the position of all points of the object. The configuration of a rigid two-dimensional object, for example, can be specified by two displacements and an angle, that of a rigid three-dimensional object by three displacements and three angles, and that of a robot arm by its joint angles. For concreteness, we will be dealing exclusively with Cartesian configurations, e.g., (x, y, θ) for objects in the plane, and not joint angle configurations. The space of all possible configurations for an object is known as the configuration space *(C-space)* of that object [Lozano-Pérez 81, 83]. An object A is represented as a point in its C-space; the coordinates of that point are the configuration parameters of A.

Stationary obstacles in the environment of a moving object A can be mapped into the configuration space of A. The resulting *C-space obstacles* are those configurations of A which would lead to collisions between A and the obstacles. Configurations on the surface of the C-space obstacle due to B are those where some surface of A is just touching a surface of B. If A and B are both three-dimensional polyhedra, the surfaces of the C-space obstacle for B arise from each of the feasible contacts between the vertices, edges, and faces of A and B (see figure 10) [Lozano-Pérez 83]. Therefore, each face of a C-space obstacle represents a particular type of geometric constraint on A. A range of positions (and orientations) of A can be represented as a volume in the C-space of A and a motion of A is a curve in the C-space.

As an illustration of the use of C-space surfaces, consider the familiar two-dimensional peg-in-hole problem from figure 6. We can construct a three-dimensional C-space of (x, y, θ) configurations of the peg. In this space, the hole defines an obstacle (see figure 11(a)). Note that although the resulting surfaces are curved, for each value of θ the (x, y) cross section of the C-space surfaces is polygonal. The surfaces represent one-point contacts and the edges at the intersections of surfaces represent two-point contacts. Line-line contacts also give rise to edges at the intersections of one-point contact surfaces. Figure 11(b) shows cross sections for a peg and chamfered hole.

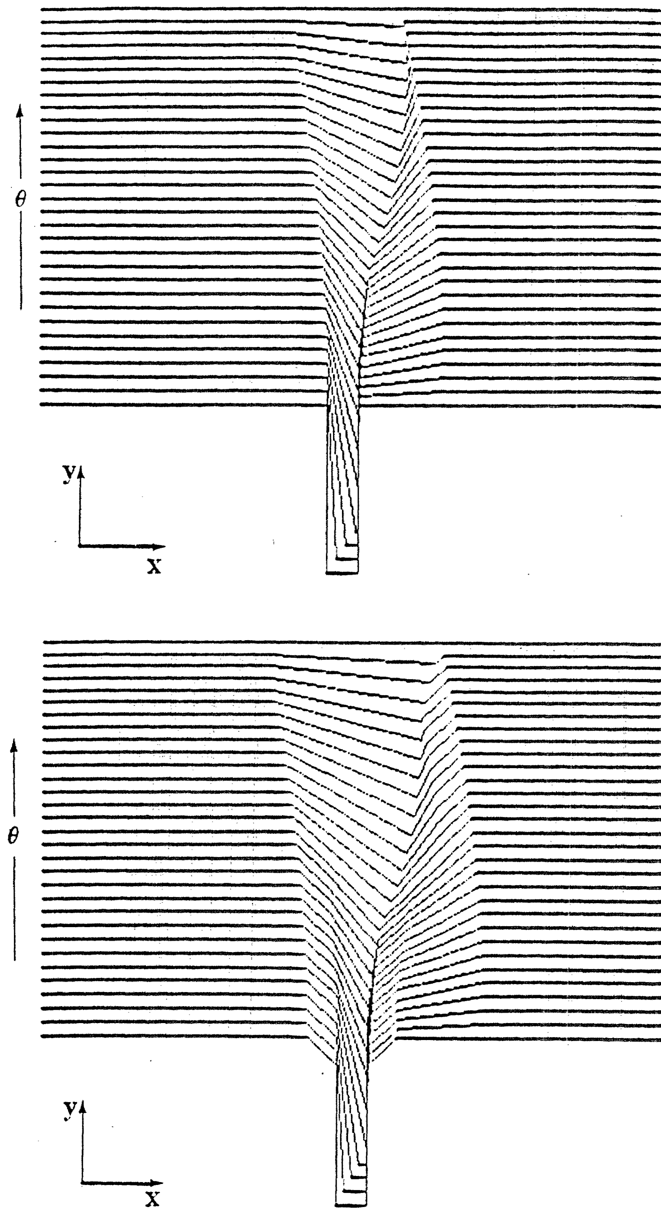

Figure 11. Cross sections of peg-in-hole C-surfaces: (a) no chamfer (b) chamfer.

The C-space representation can be extended to more general kinematic situations. In general, motion subject to geometric and kinematic constraints can be defined as collections of equalities and inequalities that must hold among the parameters that determine the configurations of the robot and the objects in the task. These inequalities represent C-surfaces [Mason 81]. Take the constraint that a robot hand remain in contact with a crank handle as it rotates. The constraint relating the position of the hand, (x, y), to the position of the crank (a constant) and its current angle, α, is a curve (one-dimensional surface) in the configuration space of the task, i.e., the (x, y, α) space.