A Fine-Motion Planning Algorithm

Matthew A. Turk

Martin Marietta Denver Aerospace
Advanced Automation Technology Section
P.O. Box 179, M.S. 0570
Denver, CO 80201

Abstract

Assembly tasks typically involve tight fits and precise positioning beyond the capability of position-controlled robots. The motions required for such tasks, called fine-motions, may be performed by means other than pure position control. Fine-motions must overcome the inherent uncertainty in the robot's position relative to its environment. This uncertainty results from errors in sensing, modeling, and control. If bounds on these errors are known or can be estimated accurately, motions may be planned that will perform fine-motion tasks despite the uncertainty. Automatic planning of robot motions to perform these tasks prevents the tedious, error-prone process of constructing a plan for each task by hand. This paper presents an approach to fine-motion planning and an implementation of the algorithms in a planning system. A simple method is explored to plan and perform assembly tasks in the presence of uncertainty using a geometric model of the task, error bounds, and a model of compliant behavior. The principal ingredients of the method are algorithms that manipulate graph representations of the task to find, and choose from, a small number of alternative plans. The approach is implemented in a system that generates plans from task descriptions of two degree-of-freedom operations.

Introduction

The use of robots in tasks such as assembly which involve tight fits and precise positioning is presently limited by the large uncertainty in the robot's position relative to its environment. This uncertainty is the result of errors in sensing, modeling, and control. If bounds on the errors are known, motions can often be planned that will perform such tasks despite the uncertainty. Automatic planning of robot motions to perform these tasks prevents the tedious, error-prone process of constructing a plan for each task by hand.

A formal approach to the automatic synthesis of fine-motion strategies is presented by Lozano-Perez, Mason, and Taylor[1]. (See also Mason[2].) They lay the framework for automatic planning and describe an approach that generates strategies directly from geometric descriptions of parts and estimates of errors. This paper complements the work of Lozano-Perez et al. by exploring an approach to fine-motion planning and implementing the algorithms in a planning system. (See also Turk[3].) The goal is to present a method of planning and performing assembly tasks in the presence of uncertainty. I have focused on two degree-of-freedom operations.

Overview of the Approach

The approach to fine-motion planning taken here involves planning a sequence of operations to reach a desired goal. Each operation is a commanded velocity that will be achieved (within some error) during free-space motion. When in contact with a surface, the compliance of the robot allows motion tangent to the surface, depending on frictional constraints.

A model of the task is specified by a higher-level process. The planning process consists of the following steps:

1. The planner first partitions the task model into regions, bounded by corners and surfaces.
2. It then calculates the range of velocity commands for each region that will guarantee travel into the next region, given error bounds and the coefficient of friction.
3. The next step is to find a combination of these velocity commands that will reach the goal. This is achieved by constructing graph representations of the task and applying algorithms to the graphs to come up with command sequences that will perform the task.
4. The planner chooses one particular sequence of velocity commands from the alternatives according to some specified criteria. This sequence of commands is the fine-motion plan.
5. Finally, it calculates the position and time information needed to decide when to execute each command. During execution, the sequencer uses this information along with the sensed position to issue each command at the proper time.

A block diagram of the system is shown in Figure 1.

The major contributions of this work are the development of the graphs to represent the task and the algorithms to manipulate these graphs. These representations and algorithms reduce the infinite number of possible compliant motion strategies to a few alternatives. These alternatives will include the plan or plans with the minimum number of commands necessary to reach the goal. The plans will work despite the presence of uncertainty.

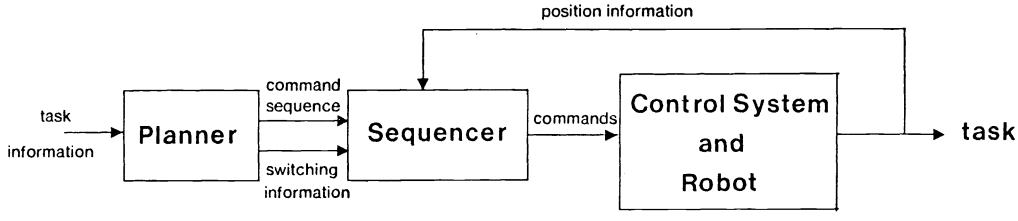

Figure 1 : Block diagram of fine-motion planning system

Fine Motions

Present-day robots are well-suited for repeatable tasks such as spray painting, spot welding, and parts transfer because these tasks primarily involve unconstrained movement in space, and they can generally be controlled by servoing the manipulator to a sequence of positions. Such motions may be referred to as "gross motions"[4]. Tasks such as assembly, however, often require small movements or relative positioning that is beyond the accuracy of available manipulators. These are called "fine motions". A simple example of a fine motion is the peg-in-hole problem of Figure 2. The insertion procedure of Figure 2(a) may fail even with a small position error, as shown in Figure 2(b).

Errors in sensing, control, and modeling complicate robot planning by introducing uncertainty in position and velocity. Uncertainty arises primarily from the following sources:

- errors in position sensing: the sensed position of the end-effector may differ from its actual position because of inaccurate sensors, errors in the kinematic model of the robot, and calibration errors.

- errors in control: the positions and velocities achieved by the control system will differ from the desired values.

- errors in the task model: the description of the task geometry and the objects to be manipulated, whether coming from a vision system, engineering drawings, or direct measurements, often includes errors.

Manipulators built to achieve the high positional accuracy necessary to perform fine motions using position control are too massive, slow, and expensive. Pure position control, then, is not sufficient to perform fine motions in the presence of uncertainty. Robots that are to perform tasks requiring fine motions need more sophisticated planning and control schemes to deal with these uncertainties.

Automatic Planning of Robot Motions

At present, instructing a robot to perform an assembly task requires one to define a sequence of operations for that task and then laboriously program a computer to perform the sequence. These activities typically involve a tedious, error prone, time-consuming effort by specially trained personnel. The goal of automatic planning is to consider the mechanics of the manipulation process, the surface interactions between objects, and the known tolerances to plan operations that work despite error in the planner's model of the task environment and inaccuracy in control of the manipulator. The motivation behind automatic planning of robot fine motions, then, is (at least) twofold: to enable the task specifications to guide the robot's motions, thus eliminating the need for an expert to program explicitly each task, and to construct robot plans that perform a task despite errors in sensing, modeling, and control. Brost[5] and Erdmann[6] give overviews of previous and current work concerned with robot motion planning dealing with uncertainty.

Models

Representing the task and the robot's interaction with that task demands models that are accurate yet not cumbersome. Tools that transform the real manipulation process into representations used in planning and guiding the process must effectively model the task geometry, errors in sensing and control, and interaction or contact between objects. This section describes the models used in the fine-motion planning algorithm.

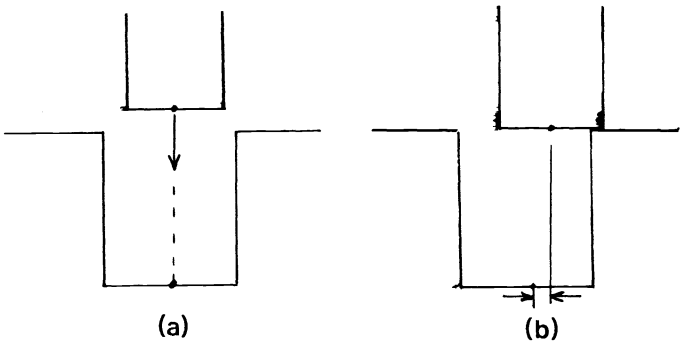

Figure 2 : Fine motion example -- peg insertion

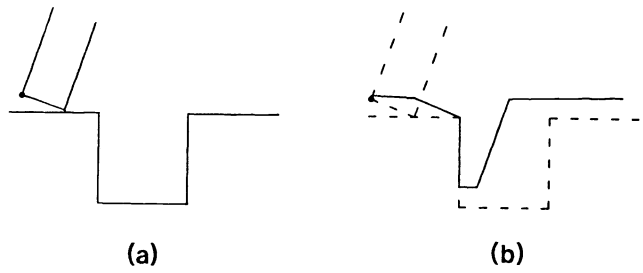

Figure 3 : C-space representation of peg-in-hole

Configuration Space

The problem of moving a rigid object among other rigid objects can be restated as an equivalent problem of moving a point among objects in a higher dimensional space, called the configuration space. C-space has been investigated at length in recent works by Lozano-Perez[7], Erdmann[6], and Donald[8]. The configuration space representation simplifies reasoning about motion of an object to motion of a point, and allows uncertainties to be represented as bounds on the position and motion of a single point. For an N degree-of-freedom robot the configuration space is N-dimensional, so a general-purpose robot manipulator will have a six dimensional configuration space. Figure 3 shows a two-dimensional C-space representation of the peg-in-hole problem for a fixed orientation.

Uncertainty

Given bounds on errors in sensing and control, these errors can be included in the representation of the task and taken into account by the fine-motion planning system. Position uncertainty, caused by inaccurate joint sensors, inaccurate modeling of the robot geometry, and calibration errors, can be represented by an N-dimensional volume element centered on the point that represents the sensed position. See Figure 4(a). The actual position of the object is then guaranteed to be within the volume element. Control uncertainty, actually error in achieving the commanded velocity, is represented as a cone centered on the commanded velocity, as in Figure 4(b). For the two degree-of-freedom case, the position uncertainty is bound by a circle of radius ε_c around the sensed position, while the angle ε_v is the velocity angle error bound. The velocity magnitude error bound is not relevant to the planning task and is not considered here. Figure 5 shows these models and possible adverse effects of the uncertainty.

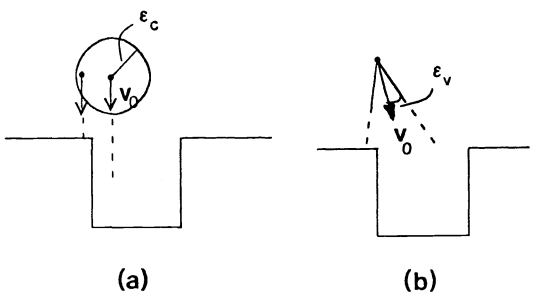

Figure 4 : Models of uncertainty

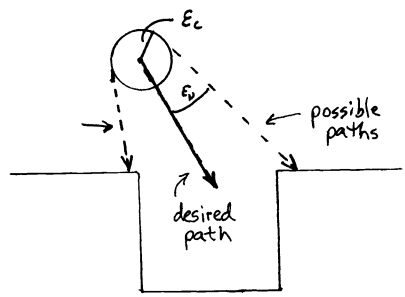

Figure 5 : Potential problem with uncertainty

Generalized Damper Model

The generalized damper model demonstrates compliant motion, similar to viscous damping, which can characterize a robot's control system. The damper is defined by the first-order relation

$$\mathbf{F} = b \, (\, \mathbf{V} - \mathbf{V}_0 \,)$$

where $\mathbf{F}$ is the force acting on the object, b is a constant, $\mathbf{V}_0$ is the commmand velocity, and $\mathbf{V}$ is the actual velocity sent to the controller. Figure 6 shows how this model is realized in the presence of surface interaction: in (a), with no perceived force, the actual (servoed) velocity is equal to the commanded velocity; (b) shows the actual velocity in the steady state assuming contact with a frictionless surface, and (c) for a rough surface. Friction serves to decrease the magnitude of the servoed velocity $\mathbf{V}$; the constant b determines how much force is applied perpendicular to the surface.

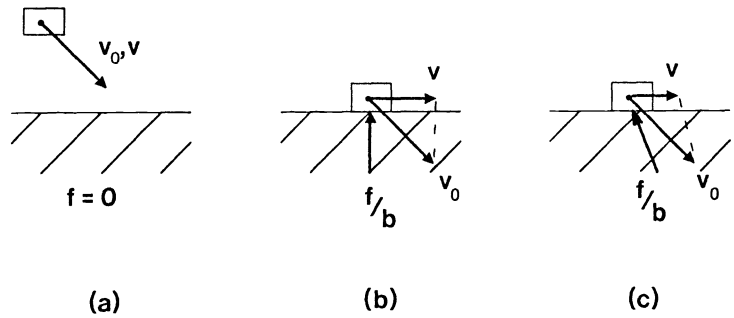

Figure 6 : Generalized damper model

Friction Cone

Interaction between surfaces is modeled using the concept of a friction cone, a simple consequence of Coulomb's law. A force directed by an object onto a surface within the cone of angle α, where α = atan μ (μ is the coefficient of friction), generates a reaction force which will halt motion of the object, while a force directed outside of the friction cone will allow movement of the object. These cases are refered to as "sticking" and "sliding", respectively. For simplicity, the distinction between static and kinetic friction is ignored.

Erdmann[6] has developed a configuration space friction cone. The compliance implemented in the generalized damper model enables a simple prediction of motion given the friction cone -- a commanded velocity within a surface's friction cone willl cause sticking on contact with that surface, wile a commanded velocity outside of the friction cone will allow sliding along the surface, i.e. compliant motion. Under a quasi-static assumption (inertial forces are assumed to be negligible[9]) the generalized damper and the friction cone completely characterize the motion of the manipulated object.

Planning the Task

This section describes the algorithm for generating a fine-motion plan from the 2D task description. This is the task of the planner in the block diagram of Figure 1. The planner takes as its input the task geometry in a configuration space representation, the error bounds ε_c and ε_v, the angle α of the friction cone, the starting configuration and the goal. Its output is a plan to perform the task, i.e. a sequence of velocity commands that will cause motion to the goal. With the given velocity commands, the robot interacts with its environment according to the generalized damper model.

Overview

The planner first divides the task geometry into a set of regions to be traversed to a goal surface or region. These regions are bounded by surfaces and corners of the configuration space obstacles. There exists a range of velocity vectors, or commands, for each region that will cause motion into the next region. The plan to perform the task consists of a combination of these velocity commands that will cause the object, a point in configuration space, to traverse the regions to the goal.

The plan is found by constructing and manipulating graph representations of the task geometry and goal. Using the models described previously, the relevant aspects of the task geometry are characterized by the **task graph**. This is a graph that represents the regions of the task and the ranges of velocity commands that will cause motion through each region. The planner applies an algorithm to the task graph that eliminates all but the most desirable combinations of commands, those with the fewest commands, to perform the task. The planner then searches the **state graph**, constructed directly from the final task graph, to find explicitly these combinations of commands, each a fine-motion plan. The advantage of these graph representations and corresponding algorithms is that the problem is reduced from finding one of an unlimited number of possible paths to the goal, to finding a combination of a finite number of velocity commands that will cause motion to the goal.

An obvious strategy to reach the goal is to choose a command to move from the starting point in region 1 to region 2, one to move from region 2 to region 3, and likewise one for each region until the goal is reached. The main problem with this strategy is the uncertainty: the position error may be large enough so that the current region is unknown. Because of this uncertainty, a command to be issued while in a particular region may be issued too soon or too late. Furthermore, a simpler plan may be appropriate: since some range of commands will often cause advancement beyond the next region, plans may be found with less than one command per region. The planner will find the plan or plans that have the fewest number of commands and are best suited to deal with the uncertainties. This is done by the algorithms that manipulate the graph representations of the task.

Associated with each sequential pair of velocity commands is a **transition zone**, the region or set of regions in which the switch from one command to the next is allowed. It is important that the transition does

not occur before reaching the transition zone, or the results will be unpredictable. Because of sensory error, however, the robot may appear to have entered the zone before it actually has. Only when the uncertainty circle has completely entered the transition region is a successful transition guaranteed by monitoring the position information. It is desirable, then, to have transition zones large enough so that the complete uncertainty circle can enter. The larger the transition zones are, the more position uncertainty can be tolerated. The planner finds strategies that have the largest possible transition zones.

The final graph, the **state graph**, directly shows a set of alternative fine-motion plans, each consisting of a sequence of velocity commands. The planner chooses one plan from the few alternatives according to some specified criteria.

Regions

The fine-motion planning strategy explored in this work is intended for assembly-type operations, which are characterized by motion constrained by objects and surfaces. The planner initially represents the geometry of the task and the robot or object to be manipulated as a sequence of regions, bounded by surfaces, corners, and other regions. The goal is a surface or a region itself. The task is then to reach the goal from the starting configuration by traversing the regions. An example of a task partitioned into regions is shown in Figure 7. The goal is to get the object, represented by a point in the configuration space, to the surface labeled "goal" from the starting position **p**.

The motivation behind breaking the task geometry up into distinct regions is to exploit the compliance of the robot to simplify the plan. Because of the compliance implemented in the generalized damper model of control, one velocity command may cause motion both in free space and while sliding along a surface. At any point there is a range of velocity commands that will cause motion to progress towards the goal. A region is a set of points in the same vicinity that have identical or similar velocity ranges. At any point in the region, a velocity command from that range will guarantee progress. Regions are particularly useful in dealing with position uncertainty. Although the exact position is unknown, it may be possible to assure that the object is in a given region, so a velocity command for that region will work despite the uncertainty.

A region is a polygonal section of the task geometry. Associated with each region is a range of velocity vectors, called T-cones (or Transition cones), that will cause motion to the next region from any point in the present region. This motion may include moving directly into the next region, sliding along a surface into the region, or a combination of free space and sliding motion.

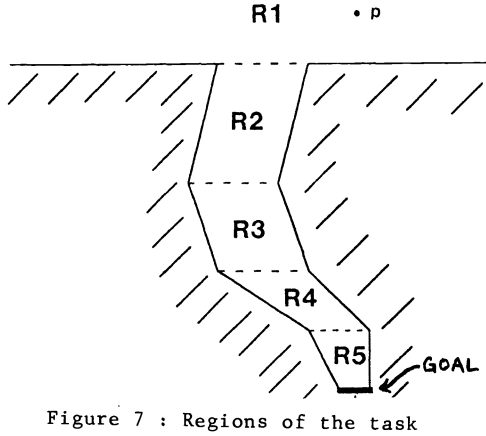

Figure 7 : Regions of the task

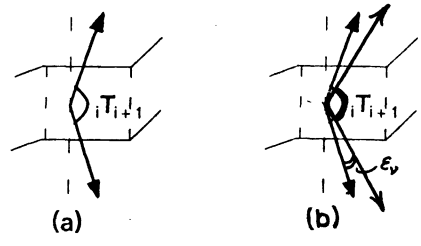

Figure 8 : T-cone with uncertainty

Calculating the T-cones

The T-cone, or Transition cone, from region i to region j, written as $_i\mathbf{T}_j$, is defined as the range of velocity vectors that will cause motion from every point in Ri to Rj. Because of the compliance implemented in the generalized damper model of control, this "forward motion" may occur in three ways: (a) motion in free space directly into the next region, (b) sliding along a surface into the region, and (c) a combination of the two. The friction cone is the limiting factor in sliding motion; the velocity command must be directed outside of the friction cone for sliding to occur. Otherwise, motion will be halted against the surface.

For a rectangular region as in Figure 8(a), the T-cone ($_i\mathbf{T}_{i+1}$) includes the range of directions between the normals of both walls, minus the friction angle on each side. Any velocity command in this range will guarantee travel from any starting position in Ri to the next region. The important element of the T-cone is that it describes forward motion for any point in the region; a command from the T-cone ensures forward motion even if the exact position of the object is unknown, as long as the object is within the region. Thus it is useful in dealing with uncertainty.

Throughout the planning stage, the T-cones are used to generate plans. Once the final plan is chosen, however, one particular command direction must be taken from each resulting T-cone to be used in executing the plan. This command direction is just the midpoint of the corresponding T-cone.

Regions With Null T-cones

Motion from open space into a hole is a special problem because the associated T-cone is null; there is no range of velocity commands that will cause motion into the whole from every point in the open space above. This is treated as a special case, with the T-cone for the transition depending on the position within the region of open space. See Turk[3] for further investigation into regions with null T-cones.

The Effect of Uncertainty on T-cones

The planner takes into account the error in command direction ε_v by limiting each T-cone by ε_v degrees all around, which in the 2D case means reducing the cone by a tatal of $2\varepsilon_v$ degrees. This ensures that the actual command direction is within the necessary T-cone. The position error ε_c does not affect the calculation of the T-cone, since the T-cone is good for any point in the region. As long as we can be sure the object is in the region, any command from the T-cone will work. Figure 8(b) shows the T-cone for a region transition with a given coefficient of friction and uncertainty.

Transition Zones

Each **transition zone** consists of a region or set of regions in which switching between successive velocity commands is allowed during execution of the plan. Consider two successive commands, $\mathbf{v}_{0_1}$ and $\mathbf{v}_{0_2}$. If $\mathbf{v}_{0_1}$ will cause motion from R3 into R6, for example, and $\mathbf{v}_{0_2}$ will cause motion from R4 into R8, then the switch between commands may occur "on the fly" anywhere in R4 or R5. (R6 is not included since the first command will cause motion to halt comewhere in that region.) Regions 4 and 5 comprise the transition zone corresponding to this pair of commands.

For a successful transition between commands to occur, the robot must be sure that the current position is in the transition zone. Because of the position uncertainty, the sensed position may inaccurately indicate that the zone has been entered. To prevent switching too soon, the robot must wait until the complete uncertainty circle is in the transition zone. Large transition zones, then, are desired to deal successfully with the position uncertainty. When a particular transition zone is not large enough to determine unambiguously a successful switch between commands, the planning system uses a timing mechanism that will allow the switch after the maximum time interval for that switch has elapsed.

Task Graph

The **task graph** is an intermediate representation of the task regions and T-cones. It represents the regions as nodes of a graph, and the T-cones between regions as arcs connecting the nodes. The creation of a task graph is the first step in finding a fine-motion plan for a task. The final task graph representation shows which combinations of velocity commands from the T-cones will perform the task, i.e. cause motion to the goal.

The initial (or nominal) task graph is composed of a node for each region of the task, including the goal, and arcs connecting adjacent regions labeled with the T-cone between the two regions. An example of a nominal task graph is shown in Figure 9. In the final task graph, as in Figure 9(b), the nodes are identical but the arcs and corresponding T-cones are different. The arcs now represent more general T-cones, velocity ranges that will cause motion not only to the next region but as far as possible. $_1\mathbf{T}_4$, for example, is found by intersecting $_1\mathbf{T}_2$, $_2\mathbf{T}_3$, and $_3\mathbf{T}_4$.

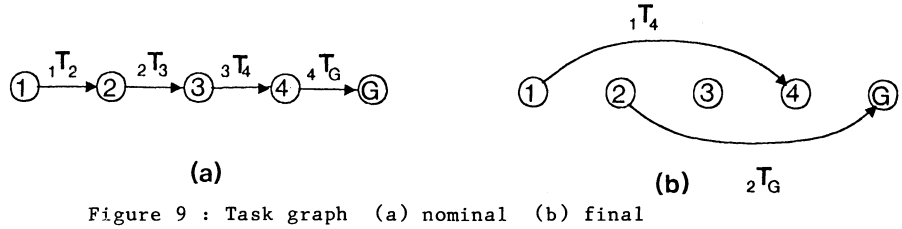

Figure 9 : Task graph (a) nominal (b) final

Consider the frist two T-cones of any nominal task graph, $_1\mathbf{T}_2$ and $_2\mathbf{T}_3$. If there is a non-zero intersection of the two, this new T-cone, $_1\mathbf{T}_3$, represents a range of commands that will cause motion from R1 to R3. A command from this range will cause forward motion when the position is in R1 or R2. Likewise, a non-zero T-cone $_2\mathbf{T}_4$ causes forward motion in R2 and R3. This pair of T-cones overlap in R2, indicating that either command will be good while in R2. Region 2 then is a transition zone because of the overlap; the second command can be issued anywhere in R2. Transition zones are determined directly from the task graph; for details see Turk[3].

Manipulation of the task graph involves intersecting T-cones and finding transition zones. We would like one velocity command to take the object as far as possible, thus decreasing the number of commands and increasing the length of transition zones. As the planner calculates the final task graph, then, it intersects adjacent T-cones to form a smaller number of "longer range" T-cones. The arcs connecting regions will be as long as possible, and the maximum overlap of arcs, representing transition zones, will occur.

The procedure to construct the task graph is presented below. Starting with a nominal task graph, the algorithm is applied to produce a final task graph. The planner uses the T-cones from the final task graph to construct a graph (called the state graph) that will make explicit the alternative fine-motion plans for the task.

Procedure to Construct the State Graph

1. Calculate the T-cone for each region and start with the "nominal" task graph.

2. Beginning with node 1, intersect the T-cones ${}_1\mathbf{T}_2$, ${}_2\mathbf{T}_3$, ... until the intersection is null. Call the last non-null intersection ${}_1\mathbf{T}_k$. This means that a command direction within ${}_1\mathbf{T}_k$ will get the object from R1 to Rk before motion halts. Connect node 1 to node k with an arc and label the arc ${}_1\mathbf{T}_k$.

3. Do the same for each successive node, but only save the resulting T-cone and corresponding arc if it reaches farther than any previous arc.

4. When the goal is reached, and the arc ${}_i\mathbf{T}_{goal}$ is labeled, the task graph is complete -- there is no need to continue for the remaining nodes.

5. Only the arcs created from this precedure are saved; the original arcs are deleted.

The State Graph

There may be many combinations of commands from the T-cones of the task graph that con complete the task, i.e. cause motion to the goal. The **state graph** is a directed graph representation that makes explicit the alternative fine-motion plans from the task graph. It is simply a method of making explicit information that is implicit in the task graph. One can look at the task graph and find combinations of its arcs that connect the first node to the goal node. Ths state graph can be searched to find these combinations.

In the state graph, the planner represents the T-cones from the task graph as nodes, or states; these states are connected by an arc if the T-cones overlap or meet at one node of the task graph. If the T-cones overlap there is a transition zone between the commands; if they meet at a node, there is a null transition zone, but switching between commands can occur after reaching the particular region. This is referred to as "sticking". A plan that includes sticking between commands will still succeed, but it will cause the robot to stop and wait for an indication that the region has been entered before the next command can be issued. This is a situation that may be avoided with sufficiently large transition zones.

To construct the state graph:

1. Create a state for each arc of the task graph, naming it with the label from the arc. The initial state comes from the arc initiating at node 1 of the task graph. The final state comes from the arc of the task graph that reaches to the goad node.

2. For every possible pair of states, ${}_j\mathbf{T}_k$ and ${}_x\mathbf{T}_y$, connect an arc from ${}_j\mathbf{T}_k$ to ${}_x\mathbf{T}_y$ if and only if $j < x \leq k$. Connect an arc from ${}_x\mathbf{T}_y$ to ${}_j\mathbf{T}_k$ if and only if $x < j \leq y$.

A path of the state graph from the first state to the goal state represents a plan with one velocity command for each node in the path. The planner searches the state graph for these paths to generate the alternative plans to complete the task.

Generating the Strategy

Once the state graph is complete, the planner generates the alternative strategies that will accomplish the task by searching the state graph for paths from the initial state to the goal. If a path consists of N nodes, then the strategy consists of N command directions, and N-1 switches must be made. The transition zones for the switches are found easily: between state ${}_a\mathbf{T}_c$ and ${}_b\mathbf{T}_d$, the transition zone is composed of regions Rb through R(c-1). A transition zone may be many regions wide, one region wide, or may be null. A null transition zone means that stopping (sticking) must occur between commands. Figure 10(c) shows alternative fine-motion plans generated from the task graph and state graph of Figure 10(a) and (b).

Executing the Task

Real-time execution of the fine-motion plan consists of issuing velocity commands in the proper sequence and time relationship. This is the task of the Sequencer in the block diagram of Figure 1. Because of the uncertainties, however, knowing when to issue each command is non-trivial. The fine-motion planning system must determine when to issue each command in the plan, given the command sequence, the task model, and the error bounds.

The plan consists of n velocity commands and the corresponding n-1 transition zones. Switching between commands must take place in the transition zones, or the results will be unpredictable: a command issued too soon may cause the robot to become stuck or to follow an incorrect path. If there were no position error,

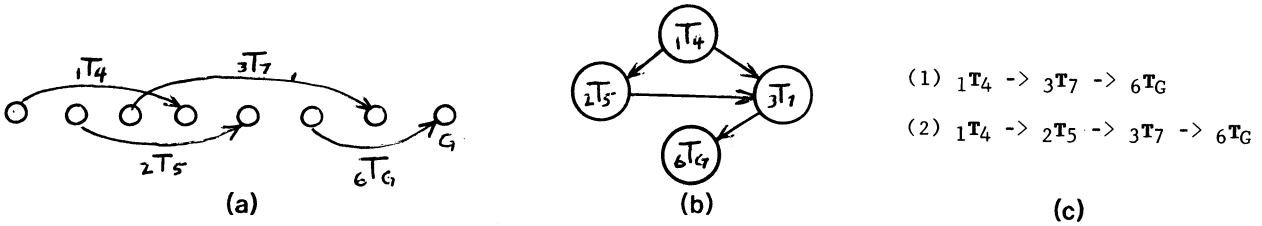

Figure 10 : Example of plan generation from the task graph

i.e. if ε_c were zero, there would be no problem -- a successful strategy would be to issue the next command at some specified position or area in the transition zone. The uncertainty in position information, however, complicates the strategy, making necessary a method of deciding when a transition zone has been entered.

The fine-motion planner uses knowledge of the position uncertainty and the geometry of the transition zone to decide when the position information unambiguously determines that the transition zone has been entered. It also uses the commanded velocity and task geometry to calculate the maximum elapsed time allowable between the issuing of a command and the entering of the transition zone.

Implementation

One purpose of this work has been to limit the domain of the fine-motion planning problem in order to explore it in greater detail. Throughout, we have introduced general concepts and then proceeded to show two degree-of-freedom figures and examples. This section briefly describes the implementation of a two degree-of-freedom fine-motion planner. The paogram PLANNER is written in C and runs on a VAX 11/780 computer interfaced with a PS2 Picture System used to display various stages of the planning.

The input data, read from a file or given by the user, consists of values for the error bounds, the coefficient of friction, the desired speed, the maximum acceleration of the robot, and the task geometry. The polygonal abstacles are described by x-y coordinates of the corner points, given in a configuration space representation. Ths planner constructs regions from the corner points of the task geometry according to a simple strategy.

PLANNER calculates a T-cone for each region, taking into account friction and uncertainties. The program constructs the task graph from the T-cones, then the state graph from the task graph, then searches the state graph using a depth-first method to find paths to the goal. Corresponding transition zones are associated with each plan. Various stages are displayed on the graphics display.

Summary

This work is intended to be a step in the progress towards automatic planning of robot tasks, focusing particularly on assembly and fine motions. For these tasks, position and control errors are significant and there is much surface interaction between the robot and other objects. The approach assumes that a detailed description of the task and error bounds are available, and that the robot can be controlled as a generalized damper. Strategies that use knowledge about the task geometry and the robot to overcome uncertainties are planned to perform a task. The basic method is structured around algorithms that manipulate graph representations of the task to develop alternative plans and then choose an acceptable fine-motion plan. The analysis is explicitly defined and the planner is implemented for the two degree-of-freedom case, but the ideas are intended to be general.

References

1. Lozano-Perez, T., M. T. Mason, and R. H. Taylor, "Automatic Synthesis of Fine-Motion Strategies for Robots", InProceedings of the First International Symposium on Robotics Research, August 1983.

2. Mason, M. T., "Automatic Planning of Fine-Motion: Correctness and Completeness", Technical Report TR-83-18, The Robotics Institute, Carnegie -Mellon University, December 1983.

3. Turk,, M. A., "A Fine-Motion Planning System for Robots", Dept. of Electrical and Computer Engineering, Carnegie-Mellon University, June 1984.

4. Whitney, D. E., "Force Feedback Control of Manipulator Fine Motions", In 1976 Joint automatic Control Conference, San Francisco, 1976.

5. Brost, R. C., "Planning Robot Grasping Motions in the Presence of Uncertainty", Technical Report TR-85-12, The Robotics Institute, Carnegie-Mellon University, July, 1985.

6. Erdmann, M. A., "On Motion Planning with Uncertainty", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, August 1984.

7. Lozano-Perez, T., "Spatial Planning: a Configuration Space Approach", IEEE Trans. Computers C-32(2), February 1983.

8. Donald, B. R., "Motion Planning with Six Degrees of Freedom", Technical Report AI-TR-791, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 1984.

9. Mason, M. T., "Manipulator Grasping and Pushing Operations", PhD Thesis, Massachesetts Institute of Technology, June 1982.