MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo No. 685 September, 1982

Symbolic Error Analysis and Robot Planning

Rodney A. Brooks

Abstract. A program to control a robot manipulator for industrial assembly operations must take into account possible errors in parts placement and tolerances of the parts themselves. Previous approaches to this problem have been to (1) engineer the situation so that the errors are small or (2) let the programmer analyze the errors and take explicit account of them. This paper gives the mathematical underpinnings for building programs (*plan checkers*) to carry out approach (2) automatically. The plan checker uses a geometric CAD-type database to infer the effects of actions and the propagation of errors. It does this symbolically rather than numerically, so that computations can be reversed and desired resultant tolerances can be used to infer required initial tolerances or the necessity for sensing. The checker modifies plans to include sensing and adds constraints to the plan which ensure that it will succeed. An implemented system is described and results of its execution are presented. The plan checker could be used as part of an automatic planning system or as an aid to a human robot programmer.

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 contract N00014-80-C-0505.

© MASSACHUSETTS INSTITUTE OF TECHNOLOGY 1982

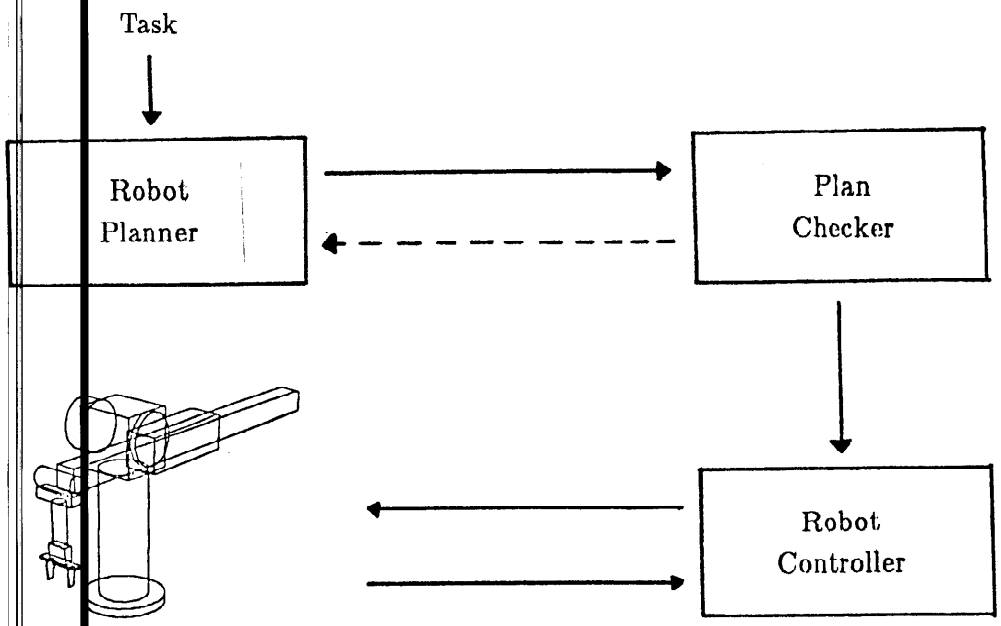

Figure 1. Robot systems considered in this paper consist of three agents. A robot controller, a plan checker and a combined robot controller and robot manipulator.

1. Introduction

This paper presents a method for checking and modifying robot plans to ensure that they will work given mechanical errors in placement and orientation of workpieces, and ranges of tolerances in the construction of the workpieces themselves. The paper goes on to suggest how the same method might be used to generate complex robot plans in the first place.

A robot plan is a program for a robot controller. It describes the motions to be made by the robot and the sense operations to carried out. It is a computer program and includes branches conditional on sense operations.

Automatic generation of plans for robots is a rich field of research that has many

unsolved problems. Very few attempts have been made to build programs which completely and automatically generate a robot plan, then command the robot to actually carry it out. Section 6 mentions some previous attempts and their successes and shortcomings. Most tasks carried out by today's robots, even in centers of Artificial Intelligence research, are planned by people.

The term *robot planner* is used to refer to an agent planning to use a robot manipulator to perform some task. A robot planner may either be a person planning the actions to be carried out by a robot, or an intelligent program performing the same task. Even for people, the task is quite difficult and it is worthwhile building automatic aids.

This paper provides the underpinnings for building programs that can automatically check whether a plan generated by a robot planner is feasible, i.e. whether it is applicable and under what circumstances it will achieve the goals set by the robot planner. Such programs are called *plan checkers*. In addition, plan checkers can at times provide information on how to fix a plan to ensure that it will work.

After a plan has been checked, a *robot controller* executes the plan, using a robot manipulator and sensors to interact with the physical world. The plan checker must reason about the capabilities of the robot controller in order to check and modify plans for the computations that the controller will make in interpreting sensor readings.

In many implementations of robot systems the distinctions between the three agents, planner, checker and controller, will be much fuzzier than shown in figure 1.

1.1 Uncertainties.

A major thrust of this work is in ensuring that a plan will succeed in spite of uncertainties in the physical world whose values can not be exactly determined at the time the plan is formulated. There are three major sources of uncertainties.

1. Robot manipulators are complex mechanical devices. There are upper bounds on speed and payload, and limits to accuracy and repeatability. The absolute positional accuracy of a manipulator is the error in that results when it is instructed to position its end effector at a specified position and orientation in space. This may depend on temperature, load, speed of movement and the particular position within its work area. Furthermore, in a situation where these parameters are all fixed, a manipulator will not in general return to precisely the same location and orientation when commanded to repeat an operation over and over. The error measures the positional repeatability. There can be contributions from stochastic effects, and from long term drift effects which can be corrected by calibration. The positional and repeatability errors of current manipulators are sufficiently large to cause problems in carrying out a large class of planned tasks in the absence of feedback during plan execution. Manipulators can be made more accurate by machining their parts more accurately and using more resilient materials. There is however a tradeoff between cost and performance of manipulators, so that there is a point of diminishing returns in trying to build ever more accurate mechanisms.

2. To make matters worse for the robot planner, multiple copies of a mechanical part are never identical in all their dimensions. It is impossible to manufacture parts with exact specifications. Instead, designers specify tolerances for lengths, diameters, and angles. Parts made from the design can take on any physical values for the parameters which fall within the designed tolerances. The effects of these variations might be large enough by themselves to be a significant factor in the success or failure of a planned robot manipulator task. When many parts are assembled into a whole, the individual small variations can combine and become large.

3. Often the most significant source of uncertainty is the position and orientation of a workpiece when it is first introduced into the task. Mechanical feeders sometimes deliver parts with large uncertainties in position and orientation, sometimes on the order of 50% of the size of the part. Conveyor belts deliver parts with even larger uncertainties. A robot planner often includes actions in the plan that are aimed at significantly reducing these initial uncertainties. The methods described in this paper can be used to analyze those

subplans also

A plan for a robot to carry out a task must take into account these three sources of uncertainty if it is to be guaranteed to succeed. There are two standard responses to the problems of uncertainties. One is to ignore them, based on the assumption that they are too small to affect the success of the plan based on nominal values. (Significant engineering may be necessary to ensure the validity of such an assumption, e.g. the construction of jigs and pallets.) A second is to estimate the uncertainties, compute the effects of uncertainties, and then decide if they are too large for the plan to work. This paper offers a third approach. It consists of computing the effects of the uncertainties symbolically. If they are small enough, the plan can be accepted. Otherwise the most significant uncertainties can be identified, and the plan can then either be constrained to succeed in spite of them, methods can be identified to reduce those uncertainties, or the plan can be modified to use a different approach which can succeed.

Throughout the paper *uncertainty* is used rather than *error*. This is intended to stress that robot plans can take into account what are traditionally called errors, and deal with them in a manner which ensures that the plans will succeed.

The theme of this paper is that while uncertain situations are hard to compute with directly, it is possible to make inferences about uncertainties and compute with those inferences.

1.2 Plans.

Minsky (1963) introduced the notion of hierarchically decomposing a plan into subplans, called *planning islands*. A plan is split up into a linear sequence of smaller subplans -- see figure 2. Each level is more detailed. The levels are often called *levels of abstraction*. The process of planning then consists of making a top-level plan less and less abstract by decomposing it into subplans, and filling in details at the lower levels.

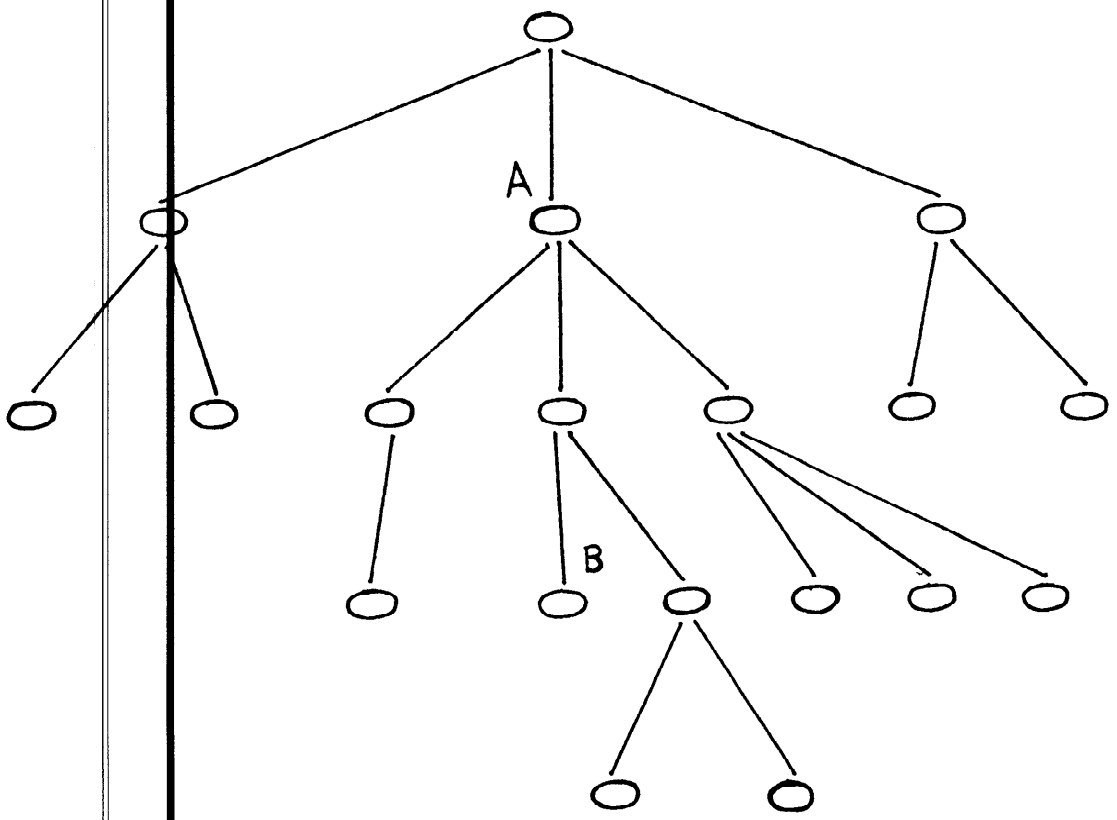

Figure 2. Plans can be decomposed hierarchically into subplans. Any particular subplan, e.g. A or B, can be abstractly viewed as a complete plan to get from the state left by its left neighbor to the state desired by its right neighbor.

There are various constraints that link the more detailed subplans together. For instance, if subplan A is followed immediately by subplan B, then at entry to subplan B the robot planner assumes the state of the world to be that upon exit from subplan A. Thus each subplan imposes constraints on subplans that follow. Dually, a subplan inherits constraints from preceeding subplans.

At any stage of a planning process there may be many decisions that have been deferred until later constraints generated by the planning process have been checked (this is commonly referred to as *posting constraints*, e.g. Sacerdoti (1977), Stefik (1981)). There is

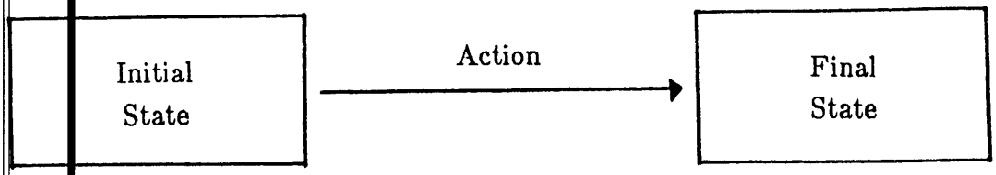

Figure 3. At any level of abstraction a plan consists of specifications of the initial and final states along with an action to effect the transition.

little advantange to making a decision until absolutely necessary. Instead, facts that affect a decision should be collected along the way and the decision should be made only when it is forced. Eventually the combined effects of the facts affecting a decision have been propagated and it simply remains to make all of the outstanding decisions while satisfying the constraints.

Each subplan can be considered at some level of abstraction to conform to the scheme of figure 3. There is a class of possible initial states of the world (which should include all states which meet the conditions imposed by the previous subplan in the chain). There is a class of possible final states of the world. The action is a mapping from initial states to final states. The possible final states should be a subset of the initial states which can be handled by the following subplan.

Figure 4 shows a plan that has been modified to include some sensing operations. The role of the robot controller is now clear. It must interpret the sensory data and relate it to parameters of the planned action.

Thus a robot planner must maintain two models. One is a model of the world. The other is a model of the state of knowledge of the robot controller (which will perhaps be itself) at the time the plan is being executed. The robot planner must reason about the interpretation capabilities of the robot controller, in terms of the knowledge it will have from sensory data and its knowledge retained from previous plan steps.

It is assumed for simplicity that a plan generated by the robot planner corresponds to figure 3. The plan may include sensing operations but they may be considered to be hidden in the black box of the initial state. The plan checker need only concern itself with sensing operations that it suggested. In reality, the robot planner and plan checker may well be highly integrated, or even indistinguishable. The essence of the model presented here nevertheless remains valid in that context.

The approach to plan checking presented here makes no particular commitment to the style of planning used to generate the plans originally. In fact section 6.2 discusses ways to use it in two different planing systems working on the same problem. The mathematics and semantics of plans are more central concerns.

1.3 Fixing Plans.

When a plan checker is checking a subplan, there are a number of possible outcomes. The following six are considered here.

OUTCOME 1: It can accept the plan as workable, perhaps adding further constraints.

OUTCOME 2: It can add sensing operations to provide more accurate execution-time information to the robot controller, to ensure that the plan will work.

OUTCOME 3: It can change the pre-conditions it requires from previous subplans in order to ensure that the plan works.

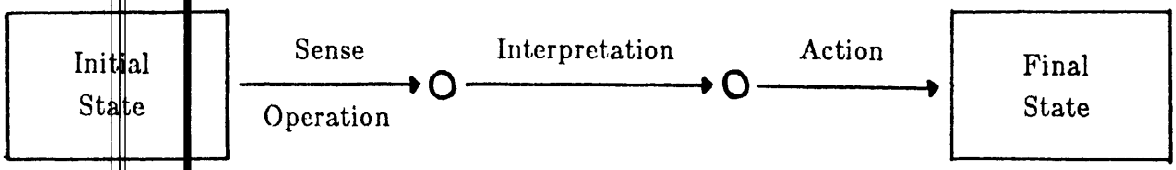

Figure 4. A plan can include an explicit sensing phase. This does not change the initial state of the world, but it does change the robot's knowledge of the state of the world. The robot controller must do some reasoning at plan execution time to interpret the sensory data.

OUTCOME 4: It can do any of 1, 2, or 3 with the additional caveat that the final state will not meet constraints as tight as those currently required by the next subplan in the chain.

OUTCOME 5: It can reject the plan if it determines that there are no sensing operations available that are powerful enough to guarantee that the action will be applicable.

OUTCOME 6: It can reject the plan if it is deemed geometrically infeasible independent of how much the uncertainties in the physical system can be reduced by sensing operations.

The algorithm presented later in this paper has five of these six possible outcomes. In addition a mathematical model is developed which provides a framework for designing such algorithms given any suitable mechanism for inference on constraints.

1.4 Outline of the Paper.

The paper approaches plan checking in three ways.

1. Section 2 presents two examples of the use of symbolic algebra and reasoning to analyze uncertainties in physical situations.

The first example is a single step in a plan presented in full detail. The algebraic expressions involved have the complexity that can be expected in realistic plan checking tasks. It is due to Taylor (1976). He estimated errors in an assembly task by propagating numerical errors forward through a geometric model of a physical situation. In this paper, however, *symbolic expressions* are computed and propagated. Once the algebraic inequalities are set up, it is possible to use them in several ways. First following Taylor, the initial constraints can be fixed and the results computed. Second, the desired results can be fixed and the constraints those results imply for the initial situation can be computed.

The second example is a simplified version of an assembly operation carried out by Taylor (Albus and Evans (1976) have a series of photographs). It includes the interactions of four plan islands and is intended to demonstrate realistic interactions which can occur between steps in a plan. The algebraic expressions are perhaps simpler than might be found in a more realistic plan. Section 2 introduces the example which is then used throughout the text to motivate and exemplify the theoretical constructs. It is simplified to bring out the essential aspects of the plan checking algorithm in such a way that the symbolic algebra can be followed without the aid of a computer.

2. Section 3 develops a formal model of plans. In addition it shows how sensing operations affect the structure of a plan. Section 4 then examines the formal mathematics of constraints within this framework, and identifies the properties of mathematical quantities associated with a plan which can be used to check the validity of the plan, suggest sensing operations, and propagate constraints forwards and backwards between planning islands.

3. Section 5 instantiates the formal model of a plan checker developed in sections 3 and 4 in terms of certain computable properties of non-linear algebraic inequalities. An algorithm exists to compute these properties. The instantiated plan checker is able to carry out precisely the computations developed in the example of section 2.

Finally section 6 relates this work to previous work on planning, and points out problems and areas for further development.

2. Some Examples

Two examples of plan checking are given in this section. The first example shows the essential features of the plan checking algorithm on a realistic single plan island. It demonstrates the basic idea of propagating desired results backwards through tolerance and uncertainty computations. The second example uses simplified geometry but shows four interacting plan islands. Only its structure is introduced in this section, along with a summary of the results of running an implemented plan checker over it. It demonstrates how a plan checker can choose the best place to introduce sensing into a sequence of operations, and how it can resolve plan decisions which may not be intuitively obvious. The example is considered in more detail in section 5 to illustrate the behavior of an implemented plan checking algorithm. The appendix shows the complete specification of the four plans as given to the implemented plan checker.

2.1 A Realistically Complex Example.

This section illustrates a plan checker symbolically analyzing the uncertainties involved in a simple insertion task. The example is taken from Taylor (1976). It contrasts the approach of propagating numeric errors with the methods described later in this paper for symbolic analysis of uncertainties. The computations for this example were carried out by the ACRONYM model-based vision system, described in in Brooks (1981a) and in more detail in Brooks (1981b).

Figure 5 is a close up view of a model of the situation described in example 2 of Appendix E of Taylor (1976).

There is a box with four holes in its top sitting on a table with a given position and orientation about the vertical axis. The position and orientation are subject to known uncertainties. A manipulator hand holding a screwdriver, with a screw at the end, is placed above one of the holes. It has three degrees of position uncertainty, and three degrees of orientation uncertainty. In addition the screw has two degrees of rotational freedom in its

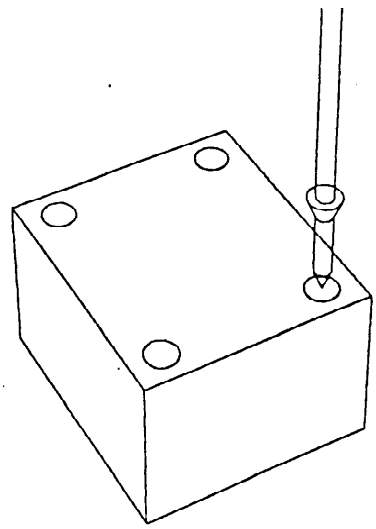

Figure 5. A box rests upon a table with an uncertain position and orientation. A manipulator hand with uncertain position and orientation grasps a screwdriver to which a screw is attached with two rotational degrees of freedom.

attachment to the screwdriver. It can wobble backwards and forwards about two orthogonal axes which go through the center of the end of the screwdriver shaft. If all the uncertainties in position and orientation are zero then the tip of the screw should be exactly in the center of the hole on the box.

Taylor used the following constraints on the errors. The variable names are mnemonics for the errors to which they refer.

$$-0.3 \leq \texttt{BOX-DELTA-POS-X} \leq 0.3$$
$$-0.2 \leq \texttt{BOX-DELTA-POS-Y} \leq 0.2$$
$$-5^\circ \leq \texttt{BOX-DELTA-ORI} \leq 5^\circ$$
$$-0.05 \leq \texttt{HAND-DELTA-POS-X} \leq 0.05$$
$$-0.05 \leq \texttt{HAND-DELTA-POS-Y} \leq 0.05$$
$$-0.25^\circ \leq \texttt{HAND-WOBBLE-X} \leq 0.25^\circ$$
$$-0.25^\circ \leq \texttt{HAND-WOBBLE-Y} \leq 0.25^\circ$$
$$-0.25^\circ \leq \texttt{HAND-WOBBLE-Z} \leq 0.25^\circ$$
$$-5^\circ \leq \texttt{SCREW-WOBBLE-Y} \leq 5^\circ$$
$$-5^\circ \leq \texttt{SCREW-WOBBLE-Z} \leq 5^\circ$$

Taylor assumed a screwdriver of length DRIVER-LENGTH that was exactly 10 inches. A fixed screw length was also assumed.

Taylor was interested in predicting the uncertainty that could be tolerated in the location of the tip of the screw relative to the center of the hole. This can be done by tracing through the coordinate transforms relating the parts of the model to get a symbolic expression for the coordinates of the screw tip in the coordinate system of the hole. The coordinate transforms can then be multiplied out symbolically (provided suitable algebraic simplification can be done – see Brooks (1981a)) to get expressions for the three coordinates. With zero errors these coordinates would be $(0, 0, 0)$. The ranges of possible values for the three coordinates indicate the position errors in the placement of the screw tip. The following is the expression so obtained for the error in the y–coordinate.

$$
\begin{aligned}
\Delta y = & -1.260 - 1.516 \times \sin(\text{BOX-DELTA-ORI}) \\
& - 1.25 \times \sin(\text{HAND-WOBBLE-Y}) \times \sin(\text{SCREW-WOBBLE-Z}) \times \sin(-\text{HAND-WOBBLE-X}) \\
& \quad \times \sin(\text{BOX-DELTA-ORI} + \text{HAND-WOBBLE-Z}) \\
& - \text{BOX-DELTA-POS-Y} \times \cos(\text{BOX-DELTA-ORI}) \\
& - \text{HAND-DELTA-POS-X} \times \sin(\text{BOX-DELTA-ORI}) \\
& + 1.25 \times \cos(\text{HAND-WOBBLE-Y}) \times \cos(\text{SCREW-WOBBLE-Z}) \times \sin(\text{SCREW-WOBBLE-Y}) \\
& \quad \times \sin(\text{BOX-DELTA-ORI} + \text{HAND-WOBBLE-Z}) \\
& + 1.25 \times \cos(\text{SCREW-WOBBLE-Y}) \times \cos(\text{SCREW-WOBBLE-Z}) \\
& \quad \times \cos(-\text{HAND-WOBBLE-X}) \times \sin(\text{HAND-WOBBLE-Y}) \\
& \quad \times \sin(\text{BOX-DELTA-ORI} + \text{HAND-WOBBLE-Z}) \\
& + 1.25 \times \cos(\text{SCREW-WOBBLE-Y}) \times \cos(\text{SCREW-WOBBLE-Z}) \\
& \quad \times \cos(\text{BOX-DELTA-ORI} + \text{HAND-WOBBLE-Z}) \times \sin(-\text{HAND-WOBBLE-X}) \\
& + 1.25 \times \cos(-\text{HAND-WOBBLE-X}) \times \cos(\text{BOX-DELTA-ORI} + \text{HAND-WOBBLE-Z}) \\
& \quad \times \sin(\text{SCREW-WOBBLE-Z}) \\
& + 1.260 \times \cos(\text{BOX-DELTA-ORI}) \\
& + \text{BOX-DELTA-POS-X} \times \sin(\text{BOX-DELTA-ORI}) \\
& + \text{DRIVER-LENGTH} \times \cos(-\text{HAND-WOBBLE-X}) \times \sin(\text{HAND-WOBBLE-Y}) \\
& \quad \times \sin(\text{BOX-DELTA-ORI} + \text{HAND-WOBBLE-Z}) \\
& + \text{DRIVER-LENGTH} \times \cos(\text{BOX-DELTA-ORI} + \text{HAND-WOBBLE-Z}) \\
& \quad \times \sin(-\text{HAND-WOBBLE-X}) \\
& + \text{HAND-DELTA-POS-Y} \times \cos(\text{BOX-DELTA-ORI})
\end{aligned}
$$

The expressions for Δx and Δz are similarly complex.

The symbolic expression bounding algorithms described in Brooks (1981a) were applied to the three coordinate expressions, given the above error bounds. The results were:

$$-0.0607 \leq \Delta x \leq 0.0510$$
$$-0.590 \leq \Delta y \leq 0.585$$
$$-0.660 \leq \Delta z \leq 0.654$$

where $-x$ is the direction down the hole (note that this is a different coordinate system to that used by Taylor). These bounds compare favorably with those obtained by Taylor:

$$-0.05 \leq \Delta x \leq 0.05$$
$$-0.54 \leq \Delta y \leq 0.54$$
$$-0.62 \leq \Delta z \leq 0.62$$

Taylor achieved smaller estimates by using more powerful numerical methods, and by ignoring some small terms.

However by carrying out the computation symbolically it is possible to answer a much richer class of questions about the geometry and the constraints. Rather than simply computing an error estimate based on propagation of errors through the physical system as above, it is possible instead to start from a desired tolerance and infer constraints on the initial situation.

Suppose, for instance, that the insertion task illustrated in figure 5 is to be achieved by applying a downward force, compliant about the screw tip, using either a passive compliance device (e.g. Drake (1977)) or active dynamic control (e.g. Salisbury (1980)). Then it is sufficient that the tip of the screw falls somewhere in the top of the open hole. Note that the hole opening is the size of the head of the screw rather than the size of the screw shaft. Compliant motion will guide the screw into its correctly seated position. This constraint can be expressed by

$$\sqrt{(\Delta y)^2 + (\Delta z)^2} \leq 0.25$$

where the hole opening has radius 0.25 inches. To simplify the algebra slightly so that the algorithms described in Brooks (1981a) can handle it, and since the errors in the y and

z coordinate are essentially independent, it is sufficient in this case to approximate the above constraint with the following two:

$$-0.25\sqrt{0.5} \leq \Delta y \leq 0.25\sqrt{0.5}$$
$$-0.25\sqrt{0.5} \leq \Delta z \leq 0.25\sqrt{0.5}$$

These constraints ensure that the action will succeed. They can be used to check the plan for the insertion task in any well characterized circumstances.

The following example uses a tighter set of constraints than those used by Taylor on errors in placement of the box. The box placement errors he used tend to swamp any other errors. A smaller amount of wobble in the attachement of the screw to the screwdriver is also assumed. The following bounds on the errors are assumed:

$$-0.05 \leq \texttt{BOX-DELTA-POS-X} \leq 0.05$$
$$-0.05 \leq \texttt{BOX-DELTA-POS-Y} \leq 0.05$$
$$-0.5^\circ \leq \texttt{BOX-DELTA-ORI} \leq 0.5^\circ$$
$$-0.05 \leq \texttt{HAND-DELTA-POS-X} \leq 0.05$$
$$-0.05 \leq \texttt{HAND-DELTA-POS-Y} \leq 0.05$$
$$-0.25^\circ \leq \texttt{HAND-WOBBLE-X} \leq 0.25^\circ$$
$$-0.25^\circ \leq \texttt{HAND-WOBBLE-Y} \leq 0.25^\circ$$
$$-0.25^\circ \leq \texttt{HAND-WOBBLE-Z} \leq 0.25^\circ$$
$$-2^\circ \leq \texttt{SCREW-WOBBLE-Y} \leq 2^\circ$$
$$-2^\circ \leq \texttt{SCREW-WOBBLE-Z} \leq 2^\circ$$

It is further presumed that the screwdriver length is not pre-determined — i.e. there are a number of screwdrivers of different lengths available for use. The plan generated must include selection of one for this task, however it is known in advance that

$$\texttt{DRIVER-LENGTH} \geq 0.0.$$

From the bounds on the errors and the expressions for Δx, Δy (such as above) and Δz it is possible to deduce bounds on those expressions in terms of the undetermined variable

DRIVER-LENGTH. For instance

$$-0.164 - 0.004420 \times \text{DRIVER-LENGTH} \leq \Delta y \leq 0.164 + 0.004420 \times \text{DRIVER-LENGTH}$$

can be deduced.

The desired constraints on Δy and Δz can then be applied to these bounds. Thus

$$\begin{aligned} -0.25\sqrt{0.5} &\leq -0.164 - 0.004420 \times \text{DRIVER-LENGTH} \\ 0.164 + 0.004420 \times \text{DRIVER-LENGTH} &\leq 0.25\sqrt{0.5} \\ -0.25\sqrt{0.5} &\leq -0.162 - 0.004204 \times \text{DRIVER-LENGTH} \\ 0.162 + 0.004204 \times \text{DRIVER-LENGTH} &\leq 0.25\sqrt{0.5} \end{aligned}$$

are sufficient to guarantee that the insertion strategy will not fail due to the screwdriver tip being outside of the boundary of the hole. These inequalities are satisfied whenever

$$\text{DRIVER-LENGTH} \leq 2.92.$$

Thus a symbolic analysis of the uncertainties in the positions and orientations of workparts and the robot manipulator has provided a constraint on the tool to be used to achieve the desired goal.

2.2 Simplified Coupled Plans.

In this section four coupled planning islands are considered. A plan checker propagates the effects of actions from one island to the next, checking whether the errors accumulate to such a degree that planned actions are untenable. When this happens it chooses the best place in the sequence of steps to introduce a sensing operation. At the same time it constrains unresolved decisions within the plan framework.

The computations for these examples were carried out by an improved version of the constraint system described in Brooks (1981a). The new version can handle explcit disjunctions of constraints (whereas the old version dealt only with conjunctions). Coupling plans often introduces disjunctions. As a by-product the new version is also able to handle quadratic forms better than the old.

Consider figure 6. A box has previously been put on a table by a two link manipulator. The task is for the manipulator to place a lid on top of the box then insert a bolt. The only source of uncertainty considered is the inaccuracy in the joints of the manipulator. A visual position sensor is available and is subject to error.

2.2.1 Geometry of the objects and sensors.

The manipulator has two links and two parallel revolute joints. It is thus restricted to work in a single vertical plane. Each link is 24 inches long. Each joint can be positioned to within $\pm 0.1^\circ$. The plans below only require that it operate at approximately the height of its first joint, and at a range of 12 to 36 inches from that joint. Using the coordinate system shown in figure 6 the uncertainty Δx in the x direction of its end-effector when nominally at coordinates $(x, 0)$ can be bounded by

$$\begin{aligned} \max(0.0002215x - 0.043262, 0.0009857x - 0.063329) &\leq \Delta x \\ \min(0.043262 - 0.0002253x, 0.063329 - 0.0009895x) &\geq \Delta x \end{aligned} \tag{2.2.1}$$

over $x \in [12, 36]$. These bounds are no more than 6% larger than the actual uncertaintities over that range. Note that the position error is larger for smaller x and smaller for larger x. The y position error behaves inversely to the x error. I.e. y error is small for small x and large for large x. However in these plans only the x error is considered.

The box is 2 inches long in the direction parallel to the x-axis. Initially the box sits on the table with an x coordinate represented by the named physical quantity *box*:`position`. Since the box was placed on the table by the manipulator, the uncertainty in that position can be characterized by (2.2.1) above. The bolt hole in the box is 0.125 inches in diameter.

The lid of the box is the same size as the box. The hole through it has a 0.125 inch bore which flares to 0.25 at the top of the lid.

The bolt is 0.125 inches in diameter. The tip of the bolt narrows down to $1/32 = 0.03125$ inches diameter at the tip. Once the tip is seated in a hole the manipulator is compliant enough for the bolt to be inserted without moving the object into which it is

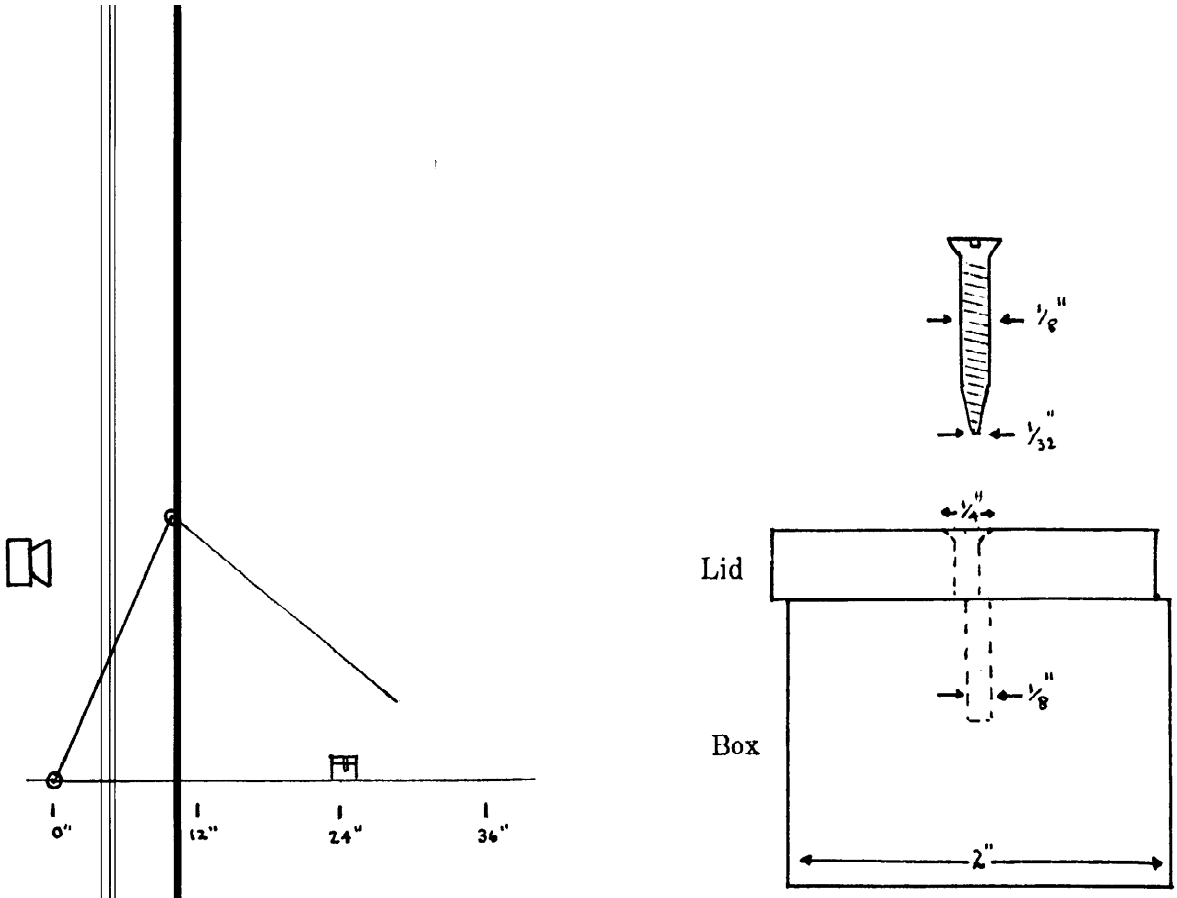

Figure 6. A two link manipulator must place a lid on a box then insert a 1/8 inch bolt (with a 1/32 inch tip) through the lid with a 1/4 inch opening and into a 1/8 inch hole in the box.

being inserted.

The visual sensor is placed directly above the base of the manipulator with a horizontal line of sight. It can measure the distance of an object on the table top by the displacement of its image from the center of the image plane. The sensor is subject to error, and the larger the distance it must measure, the larger is the error. The implemented plan checker has been run with a number of different models for the error characteristics for the sensor.

The errors are modelled by two functions: l_s and r_s dependent on the sensor reading m. Thus for a true physical value v the sensor will produce a reading m such that

$$m + l_s(m) \leq v \leq m + r_s(m).$$

An example sensor has $-l_s(m) = r_s(m) = 0.0004 \times m$, and then for physical value v the sensor is guaranteed to give a reading m where

$$m + l_s(m) = 0.9996 \times m \leq v \leq 1.0004 \times m = m + h_s(m). \qquad (2.2.2)$$

2.2.2 A four stage plan.

The plan is broken into four subplans; (A) move the lid to a position above the box, (B) put down the lid, (C) move the bolt to above the lid and (D) insert the bolt. Note that the steps to acquire the lid and bolt are ignored in this formulation. This is to simplify the problem for presentation.

The initial state of the world is determined by the position of the box. It was placed by the manipulator at *box*:position with uncertainty given by (2.2.1).

The four subplans are given in more detail. The full details of how they are presented to the implemented plan checker are given in the appendix.

Plan A. The lid is moved to a position called *lid*:position. It has the same nominal value as *box*:position. The only requirement is that the nominal position for the lid is within the workspace of the manipulator, i.e.

$$12 \leq \textit{lid}\text{:position} \leq 36.$$

The resultant position for the lid will be subject to uncertainty characterized by (2.2.1).

Plan B. The lid is released onto the box. For the subplan to work, it must be that the center of gravity of the lid is above the box. Since the box is 2 inches wide this means that

$$-1.0 \leq \textit{lid}\text{:position} - \textit{box}\text{:position} \leq 1.0\ .$$

Plan C. The bolt is moved to a position called *bolt*:position. It has the same nominal value as *lid*:position. As in plan *A* above the only requirement is that

$$12 \leq \textit{bolt}\text{:position} \leq 36.$$

The resultant position for the bolt will be subject to uncertainty characterized by (2.2.1).

Plan *D*. The bolt is inserted into the lid and through to the box. For the tip of the bolt to lie wholly within the flared hole in the lid it must be true that

$$-7/64 = -0.109375 \leq \textit{bolt}\text{:position} - \textit{lid}\text{:position} \leq 0.109375 = 7/64.$$

The bolt will comply with the hole in the lid. The condition necessary for it to be inserted into the hole in the box is then

$$-3/64 = -0.046875 \leq \textit{lid}\text{:position} - \textit{box}\text{:position} \leq 0.046875 = 3/64.$$

2.2.3 Analysis of the four plans.

These four plans will be used throughout the text to illustrate the plan checking process. In section 5 a plan checker is demonstrated analysing these four plans and their interactions. A summary is given here.

The plan checker follows through plan *A* through to plan *D* before it encounters a problem. In plan *D* the bolt tip can inserted into the lid without the aid of sensing, but relative uncertainties in the positions of the box and lid have built up so much that there is no guarantee that they will line up well enough for the bolt to slide though the hole in the lid, into the hole in the box.

The plan checker identifies the troublesome uncertainties as those for the box and lid positions but recognizes that the bolt's uncertainty is not an issue. It briefly considers sensing both the box and lid positions, but without doing any uncertainty analysis it realizes that such sensing will not change the geometric possibility of misalignment. The check starts to back up through the plans propagating back the information that the uncertainty in the box and lid positions seem to be the cause of the problems.

At plan *C* the checker considers sensing the lid position before choosing a nominal position for the bolt. It propagates the new uncertainties through to plan *D* but finds again

that the fundamental problem has not been changed. The checker resumes the backing up operation.

At plan *B* it decides that there is nothing to be sensed which will be any different from that already tried in plan *C*. It continues to back up and gets to plan *A*. It decides that reducing the uncertainty in *box*:position may suffice. It introduces the use of the visual sensor for the box position in plan *A*, then propagates the effects forward through the plans. Now the nominal values for *lid*:position and *bolt*:position depend on the sensed value for *box*:position rather than the *a priori* value.

Assume that a sensor with error characteristics given by equation (2.2.2) is used.

When the checker returns to plan *D* it finds that the plan will succeed so long as the initial box position is around either end of the range of [12, 36]. If the box is placed at the low end of this range then the sensor accuracy is high and even with an inaccurate manipulator the lid can be placed sufficiently well on the box for the holes to align. If the box is placed at the high ned of the range, then the sensor accuracy will be lower, but the extra error so introduced will be compensated for by the increased horizontal position accuracy of the manipulator in that range. If the box is placed in the middle of the range then it turns out that the accuracies of both the manipulator and the sensor are sufficiently bad so that the holes in the box and lid can not be guaranteed to be aligned sufficiently.[1]

[1]This result surprised the author. In thinking about it qualitatively before running the example through the plan checker it had seemed that the any restrictions on where the box should be placed would be of a form that constrained it to the middle of the range [12, 36]. The reasoning was that here neither the sensor nor the manipulator would be sufficiently inaccurate to cause the plan to fail. However for that to happen the shape of the sensor error characteristics must be somewhat different.

3. A Model of Plans and Sensors

The basic model of a plan used in this paper is that there is an initial state, a final state and a plan of action to change the initial state into the final state. There are three refinements to this basic model. There may be uncertainty in the initial state, there may be conditions on the applicability of the action, and the planner may generate a plan with an uncertain final state. All these need to be quantified. Furthermore the plan may be just a small part of larger plan – a planning island. Decisions concerning certain details associated with the plan may have to be deferred until adjacent plans in the plan island space have been finalized. Alternatively, the decisions required for the various islands may be mutually dependent.

There may be uncertainties in the robot planner's knowledge of the initial state, so the plan must be able to handle a set of initial states. For instance in the example used in section 2.1 the set of initial states was all possible combinations of the block's position and orientation on the table, the hand's position and orientation, and orientation of the screw on the tip of the screwdriver, subject to the constraints given.

The action may be applicable over a class of states. For instance the screw in the example of section 2 could be inserted as long as the tip was somewhere within the circumference of the hole. Given the uncertainties in the initial state of the world, the robot planner must determine whether the desired action is aplicable.

There may be a range of final states of the world that are acceptable. For instance, if the task is simply to throw a part in a bin then the particular position and orientation of the part is not important. It suffices that it is somewhere within the confines of the bin.

Planning islands provide a hierarchical decomposition of large plans into smaller plans at more detailed levels of abstraction. The model used for a plan in this paper deals with single islands at a single level of representation. Section 6 discusses linking such plan islands in the context of constraint satisfaction as introduced here. A sequel to this paper

will elaborate on these methods.

3.1 Notation.

To formalize the model of a plan it is necessary to introduce some notation. The notation introduced in this section is sufficient to follow up to section 5 when more will be necessary.

Individual functions are written as words such as *DECIDE* or *support*. Upper case functions are those for which there exists a program to compute their values. Lower case functions are mathematical entities which may not be computable.

Single upper case letters, perhaps with subscripts, such as P or C_A refer to sets. Lower case letters, such as e are used to refer to mathematical individuals. Strings of letters set in upper case typewriter font, such as BOX-POSITION-X are formal variables of a plan. Strings in lower case such as *box*:length refer to slots in the geometric model of the world. Such slots represent actual physical quantities.

Functions defined on a domain are sometimes applied to a subset of that domain, meaning the image of the subset under the function.

Expressions are constructed from constants, formal variables and slot names. *Constraints* are first order sentences over boolean-valued predicates applied to one or more expressions. Thus

$$3.0 + \texttt{BOX-POSITION-X} + \texttt{BOX-POSITION-DELTA-X}$$

is an expression while

$$-0.03 \leq \texttt{BOX-POSITION-DELTA-X} \times \cos(\texttt{BOX-DELTA-ORI}) \leq 0.03$$

is a constraint.

A set of constraints is written as a subscripted C, such as C_A and C_I. The subscript identifies a particular constraint set. A set of constraints is equivalent to the single constraint which is the conjunction of its members.

DEFINITION: The support of an expression is the set of atomic symbols which appear in it. It is written *support*(*e*). Similarly *support*(*c*) is written for the union of atomic symbols which occur in all the expressions in the constraint *c*, and *support*(C) for the union of the supports of all the constraints in the constraint set C. ∎

For example the support of the expression above is the set

{BOX-POSITION-X, BOX-POSITION-DELTA-X}.

DEFINITION: The range of a formal variable v is denoted *range*(v) and is the set of values that can be substituted for the variable. ∎

Typically, the range of a variable will be the real numbers.

DEFINITION: A set of variables V defines a space, denoted *space*(V), which is the cross product of the ranges of the elements of V. I.e.

$$space(V) = \prod_{v \in V} range(v)$$

The ordering of the product is arbitrary but fixed for index sets V. ∎

A point $p \in space(V)$ can be written $(p_{v_1}, p_{v_2}, \ldots, p_{v_n})$, where the product forming *space*(V) is ordered $v_1, v_2, \ldots, v_n$. Given $p \in space(V)$ and $v \in V$ the vth ordinate of p is written p_v and of course $p_v \in range(v)$.

Given two variable sets W and V where $W \subseteq V$, *space*(W) is identified with the appropriate natural subspace of *space*(V).

DEFINITION: Given variable sets W and V, where $W \subseteq V$, and a subset $S \subseteq space(V)$, then $proj(V, W, S)$ is the projection of S into $space(W)$. I.e.

$$proj(V, W, S) = \{ q \in space(W) \mid \exists p \in S, \forall w \in W, p_w = q_w \}.\blacksquare$$

As an example a spherical volume in three space projects into a filled circle in two space – thus

$$proj(\{x, y, z\}, \{x, y\}, \{(x, y, z) \mid x^2 + y^2 + z^2 \leq 1\}) = \{(x, y) \mid x^2 + y^2 \leq 1\}.$$

DEFINITION: Given variable sets W and V where $W \subseteq V$, and a subset $R \subseteq space(W)$, then $lift(V, W, R)$ is the largest subset of $space(V)$ which projects into R. I.e.

$$lift(V, W, R) = \{ p \in space(V) \mid \exists q \in space(W), \forall w \in W, p_w = q_w \}.$$

In particular

$$proj(V, W, lift(V, W, R)) = R.\blacksquare$$

DEFINITION: Given an expression e and a variable set V where $support(e) \subseteq V$, and $p \in space(V)$ then $[e]_V(p)$ is the interpretation of the expression e at the point p. Its value is the result of evaluating e with the substitution of p_v for v throughout for each variable $v \in support(e)$. The definition has a natural extension to the interpretation of a constraint c. Thus $[c]_V$ is a predicate on the domain $space(V)$. The definition can be extended to encompass partial evaluation of expressions. Thus if

$$V \subseteq support(e) = U$$

then $[e]_V(p)$ will be an expression with support in $U - V$ and will be the appropriate partial evaluation of e, i.e.

$$\forall q \in U - V, \qquad [[e]_V(p)]_{U-V}(q) = [e]_U(r)$$

where $r \in U$ such that $proj(U, V, r) = p$ and $proj(U, U - V, r) = q$.

For example

$$[\texttt{BOX-POSITION-X} + \texttt{BOX-POSITION-DELTA-X}]_{\{\texttt{BOX-POSITION-DELTA-X}\}}(0)$$

$$= \texttt{BOX-POSITION-X}.$$

DEFINITION Given a constraint c and a variable set V where $support(c) \subseteq V$, the satisfying set of c over $space(V)$, written $sat(c, V)$, is the set of all points in $space(V)$ where the constraint holds. I.e.

$$sat(c, V) = \{\, p \in space(V) \mid [c]_V(p) \,\}$$

For a constraint set C and a variable set V where $support(\mathrm{C}) \subseteq V$, the satisfying set of C over $space(V)$, written $sat(\mathrm{C}, V)$, is the set of all points in $space(V)$ where all the constraints in C hold. I.e.

$$sat(\mathrm{C}, V) = \bigcap_{c \in \mathrm{C}} sat(c, V).\blacksquare$$

Note that for a constraint set C where $support(\mathrm{C}) \subseteq W \subseteq V$ then

$$sat(\mathrm{C}, V) = lift(V, W, sat(\mathrm{C}, W)).$$

Note also that for two constraint sets, C_A and C_B, and variable set V then

$$sat(\mathrm{C}_A \cup \mathrm{C}_B, V) = sat(\mathrm{C}_A, V) \cap sat(\mathrm{C}_B, V),$$

where $support(\mathrm{C}_A) \subseteq V$ and $support(\mathrm{C}_B) \subseteq V$.

3.2 Representing uncertain physical situations.

There are two types of uncertainty which a plan checker must deal with. Plans can be made quite detailed, yet still incorporate unresolved decisions. Even at execution time the robot controller will not have exact values for physical parameters. These are both handled

by the use of formal variables to represent uncertain knowledge and constraints on formal variables what is known.

There are two distinct sorts of variables: those whose values though not known at plan time will have at least a known nominal value assigned them before execution time, and those which will not be known even at execution time.

Variables whose values are not known at plan time, but will be known at execution time are called *plan variables.*

Variables whose values will not be known even at plan execution time are called *uncertainty variables.*

3.2.1 Representing physical values.

Physical quantities are represented in geometric models by names. An *ad hoc* naming scheme is used in this paper to avoid the introduction of unnecessary machinery. Consider plan *A* in the example of section 2.2. The only physical quantities represented are *box*:position and *lid*:position. In a more realistic representation of the plan, the named physical quantities would include *box*:width, *lid*:width and *lid*:feeder-position.

A named physical quantity is represented as the sum of a plan variable and an uncertainty variable. Thus for instance in plan *A* the position of the box on the table, *box*:position, is represented by the expression

BOX-POS + BOX-UNC.

The plan variable BOX-POS whose exact value may be unknown at plan time represents the nominal value of the position of the box at plan execution time. The uncertainty variable BOX-UNC represents the uncertainty that the robot controller will have concerning the actual physical position of the box at plan execution time.

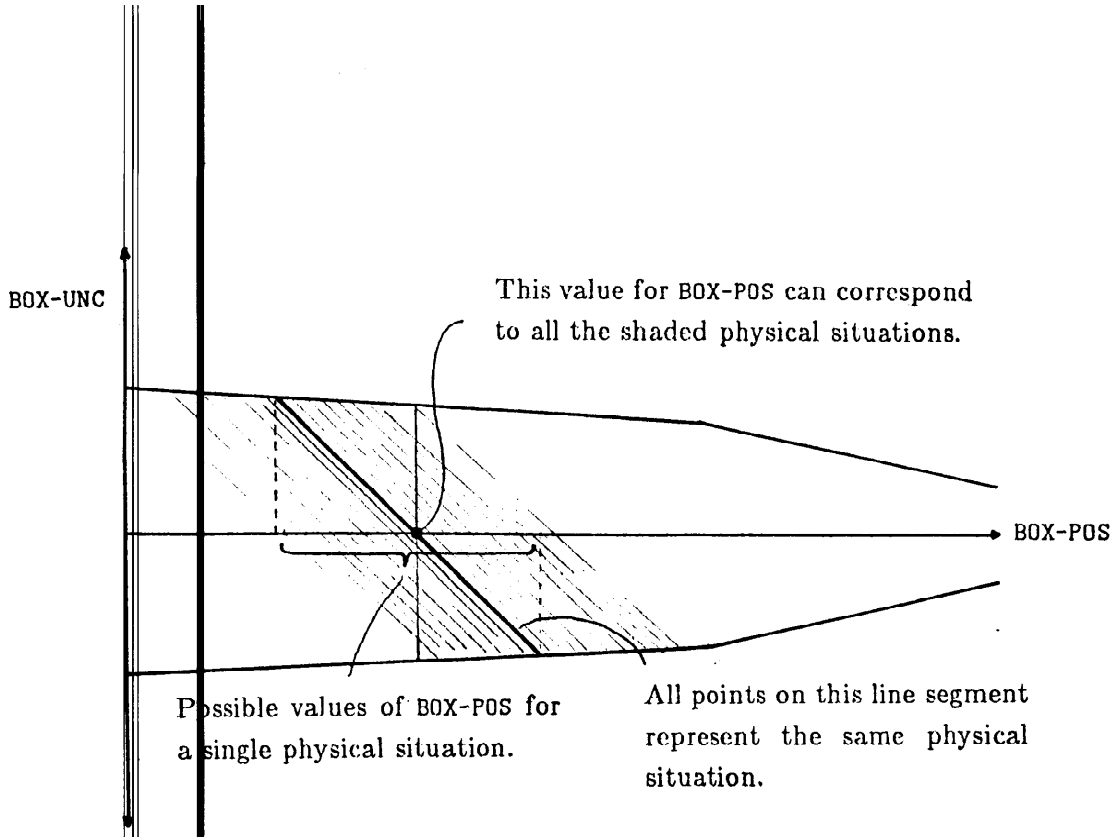

Figure 7. A given physical situation can be represented by any point on the sloped line, and hence its nominal value can take on any value in the projection onto the horizontal axis. Conversely any nominal value can correspond to any physical situation whose sloped line intersects the vertical about that nominal point.

A given physical quantity may have many different representations at plan execution time. Consider figure 7. The uncertainty variable BOX-UNC is bounded above and below by functions of the plan variable BOX-POS. A particular value for BOX-POS can model many actual physical situations – one for each value of BOX-UNC which lies within the bounds. Similarly a given physical situation can be modeled by many values for BOX-POS. Any particular physical situation corresponds to a straight line segment, with slope —1, as illustrated in figure 7. Any value for BOX-POS which lies in the project of the line segment (the intersection of the line and and the bounded region) onto the axis is a valid nominal representation of the physical situation.

3.2.2 Nominal values of expressions.

The nominal value of an expression can be recovered by substituting zero uncertainty throughout. Thus if P is the set of plan variables and U the set of uncertainties, then the

nominal value of an expressions e, where $support(e) \subseteq P \cup U$, will be given by $[e]_U(0_U)$ where 0_U is the zero point of $space(U)$.

Thus, for instance the nominal value of BOX-POS + BOX-UNC is BOX-POS. The functional form *nominal* acts on named physical quantities. Thus if *box*:position is represented as above, then

$$nominal(box\text{:position}) = \texttt{BOX-POS}.$$

This mathematical device will be useful when it is necessary to extract the nominal value of a derived expression.

3.3 What is specified and unspecified in a plan.

Plans link an initial state, an action applied to that state and a final state.

At plan time there may be unresolved decisions and so not even the nominal initial state can be known. At execution time there will be many uncertainties in the actual values of physical quantities. Thus a plan must consider a set of initial states, ranging over both unresolved decisions and physical uncertainties. With many possible initial states to consider there are many possible final states. The action of the plan thus becomes a maping from initial states to final states.

The role of a plan checker is to ensure that for all possible initial states the planned action is applicable and will lead to a final state that can be handled by the next step in the overall plan.

3.3.1 Initial state.

The insertion task in section 2.1 can be planned in some detail in terms of a nominal position for the block on the table while still representing it as a variable. Before the plan is actually executed a specific nominal position will be chosen. (Note that it may be chosen only momentarily before the plan is executed - perhaps on the basis of a sense operation.)

The uncertainty in the two coordinates of the block on the table will not be known even at plan execution time. Bounds on those uncertainties were, however, known at plan time.

The set of all *plan variables* in a plan is denoted P.

The set of all *uncertainty variables* in a plan is denoted U.

The geometry of the initial state of the world is specified in terms of the geometry (in terms of named physical quantities) of the objects in the world and in terms of expressions over constants and variables in the set $P \cup U$, representing named physical quantities. In this paper only the correspondences between named physical quantities and expressions representing them are considered. The details of representation of geometrical relations are not considered.

A set of *initial constraints*, C_I, where $support(C_I) \subseteq P \cup U$, constrains the possible initial states to which the planned action may be applied. Furthermore, $sat(C_I, P \cup U)$, those interpretations of the variables which satisfy all the constraints, can be considered as the set of possible initial states.

The set C_I is derived by the planning system by tracing through the geometry of plan islands from the given initial state of the world. Its initial derivation is not a concern of this paper.

As an example consider plan A of section 2.2. There is only one physical quantity to be represented, namely *box*:position. The initial states of the world of plan A can be represented by the sets

$$P = \{\texttt{BOX-POS}\}$$
$$U = \{\texttt{BOX-UNC}\},$$

the association

$$\textit{box}\texttt{:position} = \texttt{BOX-POS} + \texttt{BOX-UNC},$$

and the set C_I consisting of the constraints

$$12.0 \leq \texttt{BOX-POS} \leq 36.0$$
$$e_l(\texttt{BOX-POS}) \leq \texttt{BOX-UNC} \leq e_h(\texttt{BOX-POS})$$

where

$$e_l(x) = \max(0.0002215x - 0.043262, 0.0009857x - 0.063329)$$
$$e_h(x) = \min(0.043262 - 0.0002253x, 0.063329 - 0.0009895x).$$

For plan B the initial states of the world can be represented by the sets

$$P = \{\texttt{BOX-POS}\}$$
$$U = \{\texttt{BOX-UNC}, \texttt{LID-UNC}\},$$

the associations

$$box\texttt{:position} = \texttt{BOX-POS} + \texttt{BOX-UNC}$$
$$lid\texttt{:position} = \texttt{BOX-POS} + \texttt{LID-UNC}$$

and C_I consisting of the constraints

$$12.0 \leq \texttt{BOX-POS} \leq 36.0$$
$$e_l(\texttt{BOX-POS}) \leq \texttt{LID-UNC} \leq e_h(\texttt{BOX-POS})$$
$$e_l(\texttt{BOX-POS}) \leq \texttt{BOX-UNC} \leq e_h(\texttt{BOX-POS}).$$

3.3.2 Action

Associated with an action are certain applicability pre-conditions. The robot planner generates these conditions as sufficient to ensure that the geometric consequences of the action will correspond to the modelled consequences. Some of these constraints might be purely geometric. For example an insertion action requires the existence of a hole. It is assumed that the robot planner has ensured such prerequisites and can transmit the identity of such geometric features to the plan checker. In addition to the gross geometric aspects there may be certain finer details which can be conveniently expressed in terms of parameters. For instance in the insertion task of section 2.1 there was a condition that the tip of the screw lie within the circumference of the hole. This is an abstract pre-condition for the applicablity of the insertion action. In the example it was translated in two steps into conditions on the plan and uncertainty variables. Firstly it was expressed as

$$\sqrt{(\Delta x)^2 + (\Delta y)^2} \leq \texttt{HOLE-RADIUS}$$

and then as a much more complex expression as the quantities Δx and Δy were related to quantities representing the state of the world.

The plan checker model of this paper assumes that the robot planner carries out the geometric interpretation of the abstract action applicability conditions into constraints on the plan and uncertainty variables. Let that set be C_A for *applicability constraints*. Note that $support(C_A) \subseteq P \cup U$.

For plan A the abstract applicability condition is that the lid be moved to some place in the workspace of the manipulator. Since the lid is to moved to physical position *lid*:position then the condition can be expressed as

$$12.0 \leq nominal(lid\text{:position}) \leq 36.0$$

which becomes

$$12.0 \leq \texttt{LID-POS} \leq 36.0$$

which is the single member of the set C_A.

For plan B it is necessary that the center of gravity of the lid be above the extent of the box. Since the box is two inches wide, and its coordinate system has the origin in the center of the box, and similarly for the lid, the condition can be expressed as

$$-1.0 \leq lid\text{:position} - box\text{:position} \leq 1.0$$

which becomes

$$-1.0 \leq \texttt{LID-UNC} - \texttt{BOX-UNC} \leq 1.0$$

which is the single member of the set C_A.

3.3.3 Final state.

The final state of the world is similar to the initial state in that it is represented by variable sets, associations of expressions and named physical quantities and constraints on the variables.

The action of a plan transforms the initial state into the final state. Sometimes the action is modeled purely geometrically, as in the insertion example of section 2.1. Sometimes it will introduce new uncertainties into the world. Plan A moves the lid to a position with uncertainty determined by the position uncertainty of the manipulator arm.

Let V be the set of *introduced uncertainty variables*. Members of V model uncertainties, not present in the world before the application of the action, which are a result of the action itself.

The results of an action can not, in general, be modelled precisely. This is the souce of uncertainty in the plan checker's model of the world. Thus the action can not be modeled as a simple function from the set of initial states to the set of final states. Instead the geometry of the action is captured by a set $\mathrm{C}_{\mathcal{G}}$, where $support(\mathrm{C}_{\mathcal{G}}) \subseteq P \cup U \cup V$, which relates the initial state of the world to the introduced uncertainties. Thus the possible final states of the world are

$$sat(\mathrm{C}_{\mathcal{I}} \cup \mathrm{C}_{\mathcal{G}}, P \cup U \cup V).$$

If the geometry constraints correspond to a physically realizable action, then for every initial state where the action is applicable, it should be the case that there is a final state that satisfies the constraints of the geometry. Thus the geometry constraints $\mathrm{C}_{\mathcal{G}}$ must satisfy the physical realizability condition

$$proj(P \cup U \cup V, P \cup U, sat(\mathrm{C}_{\mathcal{I}} \cup \mathrm{C}_{\mathcal{A}} \cup \mathrm{C}_{\mathcal{G}}, P \cup U \cup V)) = sat(\mathrm{C}_{\mathcal{I}} \cup \mathrm{C}_{\mathcal{A}}, P \cup U) \quad \textbf{(R)}.$$

As an example the geometric constraint set $\mathrm{C}_{\mathcal{G}}$ associated with the action of plan A consists of the singleton

$$e_l(\texttt{BOX-POS}) \leq \texttt{LID-UNC} \leq e_h(\texttt{BOX-POS})$$

where the introduced uncertainty variable set is $v = \{\,\texttt{LID-UNC}\,\}$. The introduced association is that

$$lid.\texttt{position} = \texttt{BOX-POS} + \texttt{LID-UNC}.$$

3.3.4 Summary.

In summary, a plan is specified by its geometry g and by three sets of variables

P	plan variables
U	initial uncertainties
V	introduced uncertainties

subject to three sets of constraints:

C_I	initial constraints	$support(C_I) \subseteq P \cup U$
C_A	applicability constraints	$support(C_A) \subseteq P \cup U$
C_G	geometry of action	$support(C_G) \subseteq P \cup U \cup V$

A plan with geometry g is written as an 7-tuple $(g, P, U, V, C_I, C_A, C_G)$. Note that the geometry g includes the associations of named physical quantities and algebraic expressions to represent them.

3.4 Acceptable plans.

The role of a plan checker is to decide whether a plan will work and produce the desired result. The mathematical criteria for these objectives are easily stated in terms of the model developed above for plans. They are stated below. There is however some difficulty in translating these abstract criteria into algorithms. Sections 4 and 5 develop a general approach and describe a particular algorithm.

3.4.1 The action must be applicable.

The geometric constraints C_G, describing the effect of the action on the initial state of the world are valid only if the applicability conditions are satisfied. Thus to guarantee that the final state of the world meets the derived constraints it must be that the applicability conditions are satisfied for all possible initial states. Thus

$$sat(C_I, P \cup U) \subseteq sat(C_I \cup C_A, P \cup U). \qquad \textbf{(A)}$$

3.4.2 The final state must be reasonable.

The plan used for a single planning island must leave the world in a state that is feasible for the next island in the planning chain. Thus a goal condition can be formulated for a plan: given any valid initial state a plan should produce a state of the world that is a vaild initial state for the next plan.

Suppose that the next planning island in the chain has an initial state described by a set of plan variables P^*, a set of uncertainties U^* and initial constraints C_I^*.

In a simple model of planning one could demand that $P = P^*$ and $U \cup V = U^*$. The goal condition for a successful plan then becomes

$$sat(\mathrm{C}_I \cup \mathrm{C}_\mathcal{G}, P \cup U \cup V) \subseteq sat(\mathrm{C}_I^*, P^* \cup U^*) = sat(\mathrm{C}_I^*, P \cup U \cup V).$$

In a realistic planning sytem that assumption can not be made. Variables can both be introduced and removed at various points along the temporal sequence from plan island to plan island.

Variables, including plan variables, are introduced by sensing operations. Examples of this are seen below in sections 3.5 and 4.2. This type of variable introduction does not affect the goal condition statement, as the variables are not used in the description of the initial state of the plan in which they are introduced.

Variables may be introduced for a second reason. To decrease the complexity of analysis of plan islands, it might be the case that aspects of the geometry of the initial state of the world are ignored until the first plan step in which they are relevant. In this case the introduced variables must be independent of the state of the world described by the results of the previous actions. The introduction of such variables considerably complicates the statement of the goal condition, but in reality add little to its meaning, nor to the implementation of algorithms to check that the condition is met. Therefore the

remainder of this paper assumes that all aspects of the world geometry are modeled from the initial state of the world, and propagated through all actions. In an implementation of the algorithms presented later in this paper it is easy to add in this aid to efficiency.

Variables are removed when all named physical quantities with which they are associated become meaningless. For instance if an object is put down at some temporary location, then picked up, the variables which describe its temporary location have no geometric meaning in the resultant state of the world. Such variables are removed from the descriptions of the world state of all following plan islands. This convention is assumed to aid in the decoupling of planning islands. It may mean that a sequence of plans, for which a valid set of plan variables can be chosen, might be rejected due to failure to understand subtle interactions of uncertainty dependence between plan islands. This paper assumes that the increased efficiency derived from the decoupling allows extra planning effort, to find better structured plans in such obscure cases.

It can thus be assumed that $P^* \subseteq P$ and $U^* \subseteq U \cup V$. The goal condition now becomes

$$proj(P \cup U \cup V, P^* \cup U^*, sat(\mathrm{C}_\mathcal{I} \cup \mathrm{C}_\mathcal{G}, P \cup U \cup V)) \subseteq sat(\mathrm{C}^*_\mathcal{I}, P^* \cup U^*). \qquad \textbf{(G)}$$

3.4.3 Sound plans.

DEFINITION: A plan $(g, P, U, V, \mathrm{C}_\mathcal{I}, \mathrm{C}_\mathcal{A}, \mathrm{C}_\mathcal{G})$ whose following plan has initial state described by P^*, U^* and $\mathrm{C}^*_\mathcal{I}$ is called sound if both conditions **(A)** and **(G)** are true. ∎

The goal of a plan checker is to either certify that a plan is sound or modify it so that it is sound. A sound plan is illustrated in figure 8.

3.5 What a sensor is.

A sensor is used to measure a quantity in the real world. All sensors are subject to measurement errors. A plan checker must have a realistic model of a sensor and its error

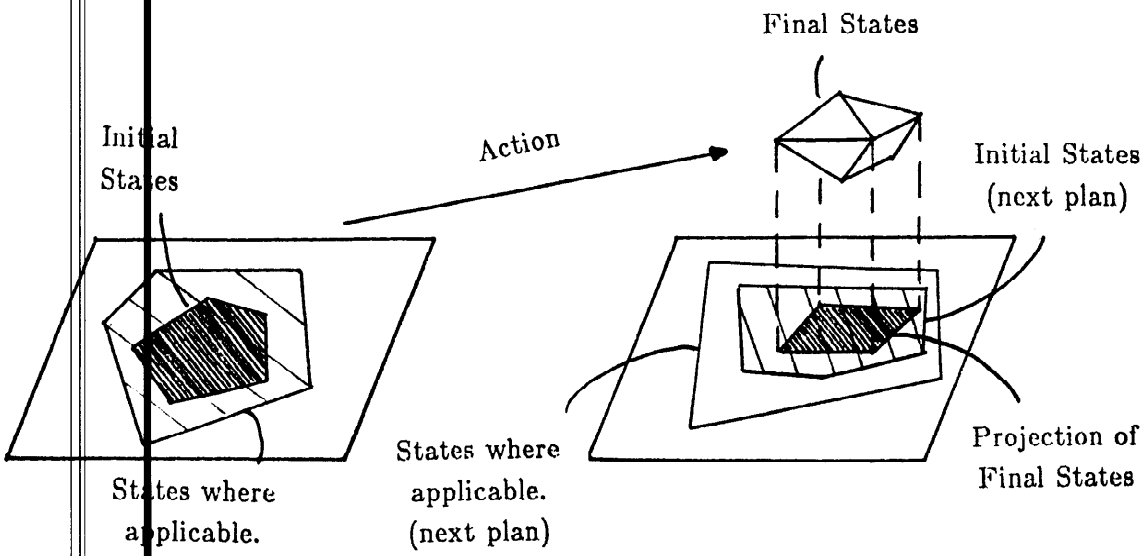

Figure 8. A sound plan has an action which is applicable for all possible initial states, and which produces a final state expected by the following plan.

characteristics in order to be able to plan the use of a sensor and to realize the consequences of such use.

3.5.1 Measurable quantities.

A quantity which can be measured must be geometrically represented as an expression in named physical quantities. In addition a sensor which can carry out the measurement must be available.

Suppose e is the expression in named physical quantities, and f the result of susbstituting in e for the representations of the named physical quantitities in a plan

$$(g, P, U, V, C_I, C_A, C_g).$$

Then $support(f) \subseteq P \cup U$. Expression f can be broken into the sum of a nominal expression

n and and uncertainty expression u by writing

$$n = [f]_U(0_U)$$
$$u = f - [f]_U(0_U).$$

Thus the uncertainty in f is zero if the uncertainties in all the named physical quantities in the support of e are also zero. Note that

$$support(n) \subseteq P$$
$$support(u) \subseteq P \cup U.$$

Typical examples of geometrically measurable quantities are the length of an object, the coordinates of the position of a detectable feature on an object, the orientation of an object or the area covered by an object on a back lighted table.

For example consider a flat rectangular object whose length, *object*:length, is represented as

$$\texttt{LENGTH} + \texttt{LENGTH-UNC}$$

and width, *object*:width, as

$$\texttt{WIDTH} + \texttt{WIDTH-UNC}$$

where LENGTH and WIDTH are plan variables, and the others uncertainty variables. Then with a suitable sensor the length of the object could be measured and in that case

$$n = \texttt{LENGTH}$$
$$u = \texttt{LENGTH-UNC}.$$

If there is a vision system available which can measure blob areas, then perhaps the area of the top of the object could be measured. Then

$$n = \texttt{LENGTH} \times \texttt{WIDTH}$$
$$u = \texttt{LENGTH} \times \texttt{WIDTH-UNC} + \texttt{WIDTH} \times \texttt{LENGTH-UNC} + \texttt{LENGTH-UNC} \times \texttt{WIDTH-UNC}.$$

3.5.2 Sensor error is a source of uncertainty.

Any sensor s capable of measuring a geometric quantity modelled by the expression $n + u$ inherently provides a measurement which is subject to error. Thus it provides a measurement which is the true value of the quantity in the real world plus some error term.

A sensor s is modelled algebraically by two error expressions, namely l_s and r_s (for left and right), where

$$support(l_s) = support(r_s) = \{\,\texttt{READING}_s\,\}.$$

A modelled sensor s can be read by evaluating READ(s) (recall that uppercase function names correspond to functions for which there is a piece of computer code somewhere). This is the nominal value returned by the sensor. If correctly modelled then the actual physical value v of what is being measured lies somewhere within an error range of this nominal value. Let

$$m = \text{READ}(s)$$

then

$$m + [l_s]_{\{\,\texttt{READING}_s\,\}}(m) \leq v \leq m + [r_s]_{\{\,\texttt{READING}_s\,\}}(m).$$

(Recall that the notation $[l_s]_{\{\,\texttt{READING}_s\,\}}$ is a λ-expression which turns l_s from an expression over one variable into a function of one argument.) Thus a sensor can have an error dependent on the value it measures. This corresponds well to most physical sensors; especially those that are non-linear. Often, of course the error expressions will be constants.

As an example, consider a sensor s which delivers a reading with $\pm 10\%$ error. Then

$$-l_s = r_s = 0.1 \times \texttt{READING}_s.$$

Consider figure figure 9. A sensor is being used to read a value for the named physical quantity *box*:position. It is represented as before by the expression

```
BOX-POS + BOX-UNC.
```

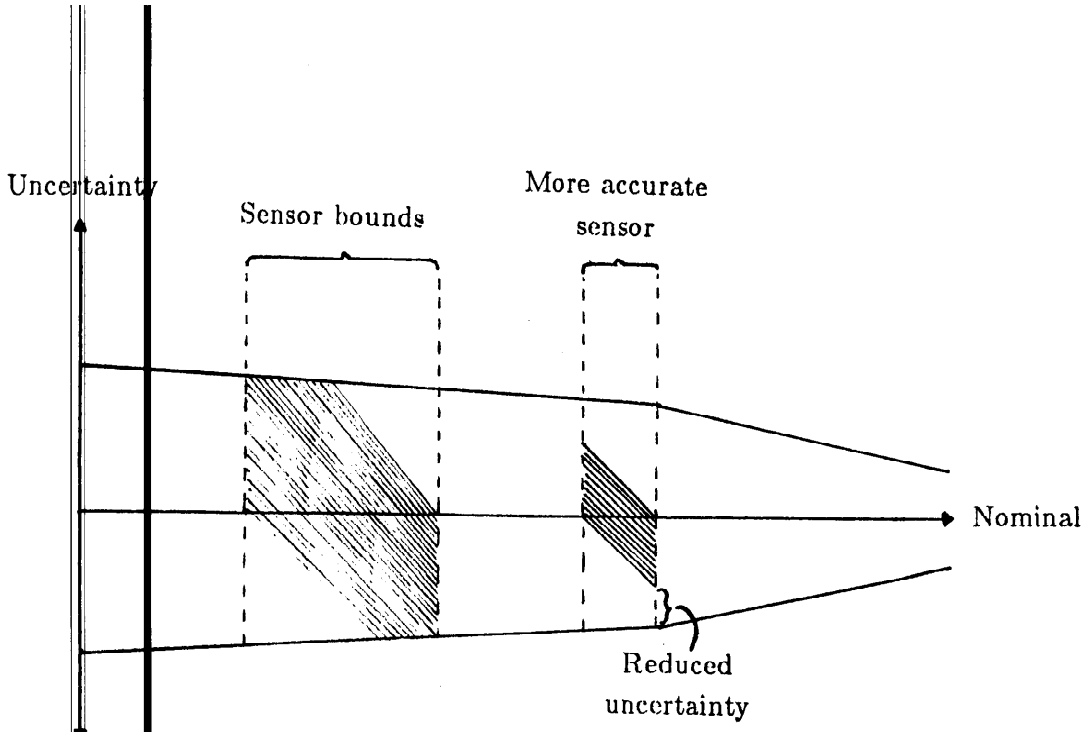

Figure 9. A sensor produces a bound on possible physical values. The correct nominal value lies somewhere in the bounded part of the plan variable axis (refer to figure 7). The shaded region gives the valid representations of the physical situation.

The sensor puts a bound on the actual value of *box*:position. Recall that a given physical value can be represented by any point on a line with slope -1 in a diagram such as figure 9. Thus all such lines which go through the bounded region of the horizontal axis comprise the set of possible realities which are consistent with the sensor reading. It makes no sense to choose a value for BOX-POS which is outside the bounded region. Thus any representation of the physical situation which is consistent with the sensor reading should be a point in the shaded region of the figure. At plan execution time the sensor reading can be used to constrain the variables BOX-POS and BOX-UNC to define a point in this region.

The sensor error expressions are used in different ways by the plan checker and the robot controller. The plan checker uses them to generate symbolic bounds on the errors that will be inherent in nominal values chosen for plan variables at execution time. The robot controller uses them to generate numeric bounds on the actual sensor readings so that it can choose a set of nominal values for the plan variables which are consistent with all sensor readings and with the state of the world known from previous steps in the plan.

4. What A Plan Checker Can Do

Given plans $\mathcal{P}$ and $\mathcal{P}^*$ where

$$\mathcal{P} = (g, P, U, V, \mathrm{C}_I, \mathrm{C}_A, \mathrm{C}_g)$$
$$\mathcal{P}^* = (g^*, P^*, U^*, V^*, \mathrm{C}_I^*, \mathrm{C}_A^*, \mathrm{C}_g^*)$$

and plan $\mathcal{P}^*$ immediately follows $\mathcal{P}$ in the planning island chain, a plan checker could decide if $\mathcal{P}$ was sound by checking whether the conditions (**A**) and (**G**) were satisfied.

Such an approach requires the comparison of satisfying sets of different sets of constraints. It can be quite difficult to explicitly compute satisfying sets whenever non-linear constraints are involved, and it is also difficult to compare them. In addition simple failure to meet one of the conditions (**A**) or (**G**) may give no hint as to how to modify the plan so that it can succeed. In general it is easier for a plan checker to modify a plan to guarantee that the modified plan is sound, rather than trying to decide whether a given plan is already sound.

The easiest way to modify a plan is to put extra constraints on the plan variables. Values must be chosen for those variables before execution time, and the plan checker can often guarantee that the plan will work by simply constraining those choices. If that fails it may be necessary to introduce sensing into the plan. Section 4.2 gives an analysis of the effects of sensing on a plan. Sensing essentially decreases the uncertainty in the world, so more constraints can be added to the initial state of the world to reflect the increased knowledge that the robot controller will have at plan execution time. Section 4.3 relates certain properties of these various sets of additional constraints to the six possible outcomes for checking a plan given in section 1.2.

4.1 Concurrently checking and rescuing a plan.

The method proposed here requires a plan checker to construct an additional set of constraints $\mathrm{C}_{\mathcal{N}}$ (the $\mathcal{N}$ is for *new* constraints), where $support(\mathrm{C}_{\mathcal{N}}) \subseteq P \cup U$, such that the

plan

$$(g, P, U, A, \mathrm{C}_I \cup \mathrm{C}_N, \mathrm{C}_A, \mathrm{C}_g)$$

is sound.

The performance of a plan checker can be measured in terms of the properties of the set C_N which it is able to compute. Some care must be taken in computing C_N to avoid *wishful thinking* where the constraints on certain of the variables are physically unrealizable. It is best to avoid making unforced decisions at any given plan island so a minimal C_N is desirable. Below a minimality condition is defined for C_N.

4.1.1 Wishful thinking.

The set C_N can not be chosen arbitrarily. It is easy to construct constraints which contain hidden wishful thoughts about the initial state of the world.

Unless some specific sensing operation is to be added to the plan, or some previous steps of the plan are to be re-planned to meet new initial uncertainty constraints, C_N should not constrain any initial uncertainties for any set of values for the planning variables P which is valid in both the old plan and the new. I.e. the following condition must be true.

$$\forall p \in proj(P \cup U, P, sat(\mathrm{C}_I \cup \mathrm{C}_N, P \cup U)), \quad lift(P \cup U, P, p) \cap sat(\mathrm{C}_I \cup \mathrm{C}_N, P \cup U) = lift(P \cup U, P, p) \cap sat(\mathrm{C}_I, P \cup U) \tag{4.1.1}$$

This condition is easily satisified when $support(\mathrm{C}_N) \subseteq P$. In fact there is an equivalent set of constraints with support contained in P whenever the above condition is satisfied.

Thus a plan checker should first try to compute a set of constraints C_N such that the new plan is sound and $support(\mathrm{C}_N) \subseteq P$. If it fails to do that, it can try to compute a set C_N where $support(\mathrm{C}_N) \subseteq P \cup U$, so long as there exist sensing operations which will enable the initial state of the world to be determined within the uncertainty requirements imposed by C_N. A prerequisite for this is that the uncertainty requirements be physically realizable.

For instance, consider the measurable quantity *box*:position from plan A represented by the expression

$$\texttt{BOX-POS} + \texttt{BOX-UNC}.$$

It may be that the constraints $C_{\mathcal{N}}$ imply that for the plan to succeed it must be the case that

$$\texttt{BOX-UNC} \in [-0.05, 0.075].$$

The desired sensing operation would be physically realizable with a sensing device having an accuracy of ± 0.05. However it is highly unlikely that

$$\texttt{BOX-UNC} \in [0.125, 0.25]$$

could ever be physically realizable. Such a range simply does not make sense as an error estimate for a sensor.

4.1.2 Minimality of additional constraints.

The new constraints $C_{\mathcal{N}}$ narrow down the set of possible initial states since

$$sat(C_{\mathcal{I}} \cup C_{\mathcal{N}}, P \cup U) \subseteq sat(C_{\mathcal{I}}, P \cup U).$$

A desirable property of the set of new constraints $C_{\mathcal{N}}$ is that it remove the minimal number of allowable initial states of the world at the same time as they ensure that the plan will work. This is equivalent to arguing for a maximal satisfying set of $C_{\mathcal{I}} \cup C_{\mathcal{N}}$. Note that this is in no sense a condition of the complexity or size of expression of the constraints, but rather on their effect. It is a desirable property because it allows maximal freedom in detailed planning concerning the other planning islands in the overall plan. It provides the least constraint on the rest of the plan.

The performance of plan checkers can be compared more formally as follows. If two plan checkers fix a plan by responding with the same outcome amongst 1 through 6 of section 1.2, but with different sets of additional constraints, $C_{\mathcal{N}_1}$ and $C_{\mathcal{N}_2}$ say, then $C_{\mathcal{N}_1}$ is

smaller (or more preferable) than C_{N_2} if

$$sat(C_I \cup C_{N_2}, P \cup U) \subseteq sat(C_I \cup C_{N_1}, P \cup U).$$

Note that not all pairs C_{N_1} and C_{N_2} are necessarily comparable.

Since outcome 1 is more preferable than outcome 6 (and in general lower numbers are more preferable than higher) a partial order on responses by a plan checker to a plan has been defined. This is however only a *partial* order and so it can not be used to determine the best response to a given plan. That depends on how the planning island to which the plan corresponds interacts with all the other planning islands in the overall plan space.

4.2 Planning to use a sensor.

Suppose the plan checker can not find a set C_N with $support(C_N) \subseteq P$ which ensures the plan will work. In principle, the plan checker should search for a set such that $support(C_N) \subseteq P \cup U$ and such that there exist sensors which can determine the state of the world at plan execution time to within the accuracy required by C_N. Sensing, however, introduces new uncertainties and new nominal values. Therefore it is necessary to restructure the representation of the plan with sensing to reflect the new variable sets.

4.2.1 New variables must be introduced.

Refer to figure 4. The world has an initial state. A sense operation is carried out. The robot controller interprets the sense operation by choosing some nominal values for plan variables, and then the plan proceeds as in the simple case of figure 3.

The goal of a sensing operation is to choose new nominal values for named physical quantities. To fit this with the variable and constraint formalism used here it is necessary to choose new names for all sensed quantities. Thus for each named physical quantity which appears in an expression describing a physical quantity to be sensed, a new plan and uncertainty variable pair must be chosen.

Suppose that in plan A it is decided to sense the physical quantity *box*:position. Originally it is represented by the expression

$$\texttt{BOX-POS} + \texttt{BOX-UNC}.$$

Let the new plan variable be BOX-SENSE-POS and let the new uncertainty variable be BOX-SENSE-UNC. The new representation of the physical quantity will be

$$\texttt{BOX-SENSE-POS} + \texttt{BOX-SENSE-UNC}.$$

Since both are representations of the same physical quantities the constraint

$$\texttt{BOX-POS} + \texttt{BOX-UNC} = \texttt{BOX-SENSE-POS} + \texttt{BOX-SENSE-UNC}$$

can be assumed even at plan time. Similar restructuring takes place when the quantity to be sensed is not a single named physical quantity, but rather derived from an expression in named physical quantities, such as the area of the top of the box in section 3.5.

4.2.2 Constraints implied by a sense operation.

Recall from section 3.5 an expression f which is the representation in terms of plan and uncertainty variables of a physical quantity to be measured, can be decomposed into the sum of a nominal expression and an uncertainty expression as

$$f = n + u.$$

Similarly let $o + v$ be the expression in the original variables which it is replacing, where o is the nominal component and v the uncertainty component. Since the old and new expressions represent the same physical quantity, the constraint

$$n + u = o + v \tag{4.3.1}$$

must hold.

A sensor reading at execution time provides a nominal value for the expression n. I.e. the robot controller can interpret a sensor reading of m by assuming the constraint

$$m + [l_s]_{\{\texttt{READING}_s\}}(m) \leq n \leq m + [r_s]_{\{\texttt{READING}_s\}}(m).$$

All such constraints will be combined (also with constraints already known) and the robot controller will choose consistent nominal values for all the plan variables P.

But why not simply use the nominal value returned by the sensor as the nominal value for expression n at plan execution time? There is a problem with consistency of multiple sensor readings. Since each individual sensor has errors in its measurement, if more than one sensor is used to measure values for non-independent expressions then the nominal sensor values will usually be inconsistent if used as the nominal values for the expressions.

Given that the robot controller will not use the nominal value returned by sensors directly, it is not possible to use the sensor error characterization directly to constrain the *a priori* runtime uncertainty of using a sensor. The analysis below proceeds as follows. First the range of possible sensor readings is characterized, then the possible interpretations of each reading are characterized as in figure 4. From that an *a priori* uncertainty can be deduced.

First consider the range of possible sensor readings which might be returned. The actual physical quantity *a priori* modelled by $o+v$ must lie in the error range sensed. Thus

$$m + [l_s]_{\{\texttt{READING}_s\}}(m) \leq o + v \leq m + [r_s]_{\{\texttt{READING}_s\}}(m). \qquad (4.3.2)$$

Let m be replaced by a new plan variable $\texttt{READING}_s$. At plan execution time an exact value can be obtained for this plan variable. Recall that l_s and r_s are expressions in $\texttt{READING}_s$. Then the above constraint becomes

$$\texttt{READING}_s + l_s \leq o + v \leq \texttt{READING}_s + r_s. \qquad (4.3.3)$$

The nominal value to be chosen for n by the robot controller ensures that

$$\texttt{READING}_s + l_s \leq n \leq \texttt{READING}_s + r_s$$

giving

$$\begin{aligned} \texttt{READING}_s + l_s &\leq \texttt{READING}_s + r_s - u \\ \texttt{READING}_s + l_s - u &\leq \texttt{READING}_s + r_s \end{aligned}$$

so that

$$l_s - r_s \leq u \leq r_s - l_s. \tag{4.3.4}$$

Recall that l_s and r_s are expressions involving $\texttt{READING}_s$.

This last constraint reduces the uncertainty u in terms of $\texttt{READING}_s$ which is itself constrained in terms of the original model by (4.3.3). Together with (4.3.1) this also constrains the possible range of n.

In the special case that the expressions l_s and r_s are independent of the variable $\texttt{READING}_s$ (e.g. for a sensor that has constant error characteristics over its whole range) there is no need to introduce $\texttt{READING}_s$ as a plan variable, and constraints (4.3.1) and (4.3.4) suffice to constrain the possible values which will be chosen for n, and for characterizing the uncertainty u.

At first sight (4.3.4) seems to underconstrain the uncertainty u. Recall, however, that it is the uncertainty at plan time in the derivation of n in terms of o, given that the plan checker has no knowledge of the algorithm the robot controller will use in determining a consistent set of nominal values. The implemented plan checker described in section 5 makes better use of the accuracy of available sensors by introducing three complications. First it assumes that all constraints are "well behaved" in some sense and that the plan checker can determine at plan time all discontinuities in the valid values for a plan variable derived from a sense operation. Secondly it plans sensing operations one at a time, with an interpretation phase interposed between any two of them. Thirdly it conservatively adds constraints to C_I which ensure that the robot controller can use the nominal value returned by the sensor as the nominal value for the physical quantity being measured.

4.2.3 The restructured plan.

Let $P^+ \supseteq P$ include introduced plan variables, and $U^+ \supseteq U$ include the introduced uncertainty variables. Also include in P^+ any variables $\texttt{READING}_s$ associated with sensor s when either l_s or r_s included it in their supports.

The new expressions associated with some of the named physical quantities in the goemetry g of a plan $(g, P, U, V, \mathrm{C}_I, \mathrm{C}_A, \mathrm{C}_G)$ make it necessary to reconstruct the constraint sets C_A and C_G. Let the new versions be C_A^+ and C_G^+. Their supports do not contain any variables from P and U associated with a named physical quantity which is in an expression whose value will be sensed. Let g^+ be the geometry g, along with the new associations of expressions and named physical quantities.

Let C_S be the set of constraints on the new variables which can be deduced at plan time. Each sensor contributes either two or three constraints. A sensor with error dependent on its reading contributes constraints (4.3.1), (4.3.3) and (4.3.4). A sensor with independent error contributes only (4.3.1) and (4.3.4). Clearly

$$support(\mathrm{C}_S) \subseteq P^+ \cup U^+.$$

The restructured part of the plan which follows sensing can now be written as

$$P^+ = (g^+, P^+, U^+, V, \mathrm{C}_I^+, \mathrm{C}_A^+, \mathrm{C}_G^+)$$

where $\mathrm{C}_I^+ = \mathrm{C}_I \cup \mathrm{C}_S$. Once constructed it can be analyzed and checked by the plan checker in almost the same manner as the original plan.

The only way in which the resulting plan must be treated differently from any other plan is that a constructed set of constraints C_N may not constrain any introduced plan variables in the set $P^+ - P$. The implemented plan checker described in section 5 actually keeps track of such variables separately, and constructs sets C_N with support in the original set P. Any constraints on the introduced variables are thus expressed in terms of constraints on the original variables. A formal notation for this representation has not been introduced, in the interests of clarity and brevity.

4.2.4 The robot controller interprets multiple sensors.

At plan execution time the robot controller makes measurements using all the

prescribed sensors. It then must interpret the values in order to choose a set of nominal values for various physical parameters, which will be used to carry out the planned actions.

The plan checker sets the stage for the execution time sensor interpretation by constructing a set of constraints which must be satisfied by the new nominal values to be chosen for plan variables in $P^+ - P$.

A slight extension of previous notation needs to be introduced. Given a set of constraints C, a set of variables A and a point $a \in space(A)$, then

$$[\mathrm{C}]_A(a)$$

is the set of constraints C partially evaluated at the point a.

Let M be the set of all formal variables $\texttt{READING}_s$, one associated with each sensor s to be used. (Note that M and P^+ may have non-empty intersection.) Let C_M be a set of constraints of the form

$$\texttt{READING}_s + l_s \leq n \leq \texttt{READING}_s + r_s.$$

There is one such constraint for each nominal expressions n being sensed by sensor s.

Now consider the situation at plan execution time. The plan variables in P already have nominal values assigned to them. Let $p \in space(P)$ represent those assignments. The sensors are all read. The result is a point $m \in M$. (Recall that the error characteristics of the sensors are encapsulated in the constraints set C_S.) The problem then is to determine a set of nominal values for the variables in $P^+ - P$, consistent with the sensor readings, the constraints derived at plan time and the nominal values already chosen for variables in P.

The robot controller can find a set of consistent nominal values for all plan variables, by choosing a point

$$p^+ \in space(P^+)$$

such that

$$proj(P^+, P, p^+) = p$$

and

$$p^+ \in proj(P^+ \cup U^+, P^+, sat([\,[\mathrm{C}_I \cup \mathrm{C}_S \cup \mathrm{C}_M]_P(p)\,]_M(m), P^+ \cup U^+)).$$

The first of these two requirements simply says that the old nominal values for variables in P are retained. The second takes into account the sensor readings and chooses an interpretation of the sensors consistent with the existing constraints.

4.2.5 Sensors can checkpoint plans.

Given that a sense operation has been introduced into a plan it is possible for the robot controller to check that the sensor readings could possibly correspond to the model of the world used by the robot planner and the plan checker. It implicitly must do this in order to interpret the sensor values and choose nominal values for the quantities being measured. I.e. it implicitly checks whether

$$sat([\,[\mathrm{C}_I \cup \mathrm{C}_S \cup \mathrm{C}_M]_P(p)\,]_M(m), P^+ \cup U^+)$$

contains a point consistent with the values for variables in P, and the sensor readings, when searching for an interpretation of the sensor values. If such a point does not exist then the measurements from the sensors are inconsistent with the model of the world.

The plan checker can arrange for the robot controller to do more however. The robot controller can be told how to to identify which sensor readings are implausible in themselves.

At plan time the plan checker can include in P^+ a variable $\texttt{READING}_s$ for each sensor s, independent of whether l_s and r_s depend on that variable, and include a constraint of the form (4.3.3) in C_S. Then it can compute a set B_s where

$$B_s \supseteq proj(P^+ \cup U^+, \{\,\texttt{READING}_s\,\}, sat(\mathrm{C}_I \cup \mathrm{C}_S, P^+ \cup U^+)).$$

Then at plan execution time when sensor s is read, and a value m_s is returned, the plan can be checked by testing whether $m_s \in B_s$. If not, then the planned model definitely does

not fit the current physical situation. Furthermore the identity of sensor s can give any error recovery system a handle on where the inconsistency lies.

4.3 How additional constraints affect the plan.

In section 1.2 six possible outcomes for a plan checking algorithm were outlined. That grouping was somewhat arbitrary but the six outcomes are maintained here to explain how different properties of a computed set C_N can be used to characterize what can be done to fix a plan. The mathematical considerations in the characterization of the set C_N are treated for the six outcomes below.

It should be emphasized the the particular set C_N computed by a plan checker depends on the plan checker itself. Plan checkers differ in the extent to which they need to alter a plan to assure themselves that it will work. Such differences can be compared if costs are ascribed to possible alterations to a plan. The issues involved will not be addressed in this paper.

4.3.1 What the plan checker does.

The plan checker is given a plan P. It computes a set C_N of additional initial constraints that it deems necessary to guarantee that the plan is sound. If $support(C_N) \subseteq P$ then it is finished and the final plan is

$$(g, P, U, V, C_I \cup C_N, C_A, C_G).$$

Otherwise the plan checker introduces sensing into the plan to derive plan P^+. It then computes a new set C_N^+ as before on the basis of P^+ resulting in a final plan

$$(g^+, P^+, U^+, V, C_I^+ \cup C_N^+, C_A^+, C_G^+).$$

4.3.2 Six outcomes.

OUTCOME 1: If no sensing was introduced, $support(C_N) \subseteq P$ and (A) and (G) hold then the new plan is sound and is simply the given plan subject to some extra constraints on the plan variables. That is to say, some of the as yet unresolved decisions concerning the plan have been further constrained.

Furthermore, if

$$C_N = \emptyset$$

or more generally

$$sat(C_I \cup C_N, P \cup U) = sat(C_I, P \cup U)$$

then the original plan is the same as the new one, so in fact the original plan was sound.

OUTCOME 2: Suppose sensing has been introduced, (A) and (G) are true for the new plan and $support(C_N^+) \subseteq P^+$. Then the plan with sensing is sound.

OUTCOME 3: Suppose sensing has been introduced, (A)and (G) are true for the new plan, but $support(C_N^+) \not\subseteq P^+$, while $support(C_N^+) \subseteq P^+ \cup U^+$. In this case plan P^+ should be rejected and the plan checker should back up to the original plan P augmented with C_N. Note that $support(C_N) \not\subseteq P$. Now the previous plan islands should be rechecked, but with $C_I \cup C_N$ as a stronger goal constraint than previously supplied. Essentially this strategy will force sensing to be carried out earlier in the overall plan so that the propagated uncertainty that reaches this plan island will be reduced.

OUTCOME 4: If any of the above conditions are met, except that (G) does not hold, then the new plan is physically realizable except that it doesn't achieve the desired goal. The set of final states which can arise should be propagated forward to the next planning island, in the hope that subsequent sensing and actions can be modified to handle the uncertainty introduced at this step of the plan.

OUTCOME 5: If none of the above conditions could be met (e.g. there are neither powerful

enough sense operations available nor could the planning islands before and ahead of this island be adapted sufficiently) then the plan $\hat{P}$ is unworkable in the context of the global plan.

OUTCOME 6: If $sat([C_I \cup C_A]_U(0_U), P) = \emptyset$ then the plan is unworkable independently of how much the uncertaintites in the physical system can be reduced by sensing operations. The test simply asks whether the constraints are satisfiable even with zero uncertainties.

5. An Algorithm For Dealing with Position Errors And Toleranced Parts

At the heart of a plan checker there must be a system which can reason about constraints, about their satisfying sets, and about the projections of those satisfying sets into subspaces.

The ACRONYM (Brooks (1981a)) system used such a constraint manipulation system to reason about consistent interpretations of image features. This section takes a constraint manipulation system of the form used by ACRONYM and constructs an algorithm for checking the algebraic aspects of robot plans. It is capable of producing five of the six outcomes previously detailed. In particular it propagates constraints forwards and backwards amongst planning islands, it introduces sensing operations when necessary and appropriate, and when a plan is rejected it gives some analysis of what is wrong with the plan. Extending it to include the sixth possible outcome (outcome 4) is straightforward, but the extra complexity detracts from the presentation of the algorithm.

Section 5.1 introduces some more notation concerning satisfying sets of sets of constraints.

Section 5.2 details the formal properties of the constraint system used in the plan checker. It is known as the SUP-INF method. These properties are precisely those needed by the plan checker developed through the rest of section 5. Any other constraint system with these properties could equally be used as the core of the plan checking algorithm. The performance of the constraint system (and hence the algorithm) is not formally addressed. The constraint system used here is understood formally when restricted to linear sets of constraints. However for non-linear constraints the best characterization known so far is informal and empirical. Section 4.3 discusses the issue of comparison of performances of plan checking algorithms.

Section 5.3 introduces an important sub-procedure. It projects a set of constraints into a subspace, over the satisfying set of a second set of constraints. This procedure is the

workhorse for constructing sets C_N.

Section 5.4 describes the main plan checking algorithm based on the SUP–INF method. Section 5.5 details how the SUP–INF method can be used to decide which physical quantities need to be sensed, and section 5.6 gives a detailed example of the plan checker on the four coupled plans introduced in section 2.2.

5.1 More notation.

The constraint methods used at the core of the plan checking algorithm described in the following sections rely on estimating bounds on expressions over satisfying sets of sets of constraints. Some additional notation is convenient in order to characterize their behavior.

DEFINITION: Given an expression e, and a set of constraints C, let $W = support(e) \cup support(C)$, let $S = sat(C, W)$ then define

$$lub(e, C) = \sup_{p \in S} [e]_W(p)$$

and

$$glb(e, C) = \inf_{p \in S} [e]_W(p).$$

Thus $lub(e, C)$ is the least upper bound on the values achieved by the expression e over the satisfying set of the constraints C. The greatest lower bound $glb(e, C)$ is defined similary.

5.2 The computational tools.

Bledsoe (1975) introduced what he called the SUP–INF method, to determine whether sets of linear equalities had integer solutions. The problems he wished to solve arise in automatic generation of proofs of correctness of programs using methods of inductive assertions. Shostak (1977) extended the method and showed that it was equivalent to the simplex method over real linear inequalities.

This author (Brooks (1981a), (1981b)) extended the SUP–INF method to handle a

large class of non-linear inequalities, and included simple extensions to handle certain elementary functions (e.g. sin and cos) in a primitive way. To support an implementation of a complete plan checker, the author had to extend the method further to handle disjunctions of both inequalitities and conjunctions of inequalities. It already implicitly handled simple conjunctions of inequalities. As a by-product of this development it became possible to include a much fuller treatment of quadratic forms.

This section describes the computable functions of the extended SUP-INF method, and characterizes their capabilities. The precise algorithms used in the earlier versions of SUP-INF method can be found in Brooks (1981a) and Brooks (1981b).

Expressions handled by the extended SUP-INF method can include operations for addition, subtraction, multiplication, division, square root, maximum, minimum, and (to a limited extent) some trigonometric functions. Operations are on numbers, the special symbols ∞ and $-\infty$ and formal variables.

Constraints are inequalities and conjunctions and disjunctions of constraints. Disjunctions can arise from simple inequalities if MIN is placed on the left of "$\leq$" or MAX on the right.

5.2.1 Bounding with projections.

The SUP-INF method takes its name from two procedures, SUP and INF which bound expressions over satisfying sets of constraint sets. In general they do not provide the best possible bounds, but in many cases (linear for instance, see below, but also often otherwise) they do.

Typically the symbolic result of SUP will be an expression of the form $\min(e_1, \ldots, e_n)$ while the result of INF will be of the form $\max(e_1, \ldots, e_n)$.

Consider as an example the set of constraints defined by

$$C_{\mathcal{E}} = \{ x \times x \leq y, x + y \leq 7 \}.$$

Then the procedures SUP and INF produce the results:

$$\text{SUP}(\text{“}x + y\text{”}, C_{\mathcal{E}}, \{ y \}) = \min(7, 2.1926 + y, y + \sqrt{y})$$

and

$$\text{INF}(\text{“}x\text{”}, C_{\mathcal{E}}, \emptyset) = -3.1926$$

The two procedures take three arguments. First consider the following special case. Let e be an expression, C a set of constraints, and suppose that $support(e) \subseteq support(C)$. Then Brooks (1981a) shows that

$$lub(e, C) \leq \text{SUP}(e, C, \emptyset)$$

and

$$glb(e, C) \geq \text{INF}(e, C, \emptyset)$$

whenever C is satisfiable, i.e. whenever

$$sat(C, support(C)) \neq \emptyset.$$

No guarantee is made concerning the results returned by SUP and INF when the constraints are not satisfiable.

For linear sets of constraints the extended versions of SUP and INF behave identically to those of Shostak (1977). He showed that under those conditions they actually find the best possible bounds. That is for a constraint set C which consists entirely of linear constraints, and a linear expression e then

$$lub(e, C) = \text{SUP}(e, C, \emptyset)$$

and

$$glb(e, C) = \text{INF}(e, C, \emptyset).$$

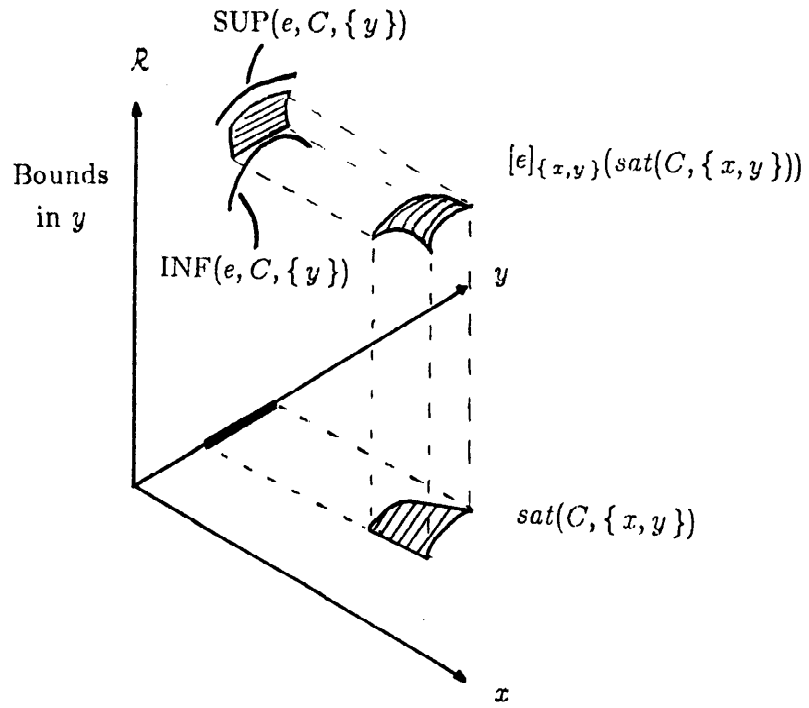

Figure 10. An illustration of the projection and bounding behavior of the procedures SUP and INF. The text contains detailed commentary.

Procedures SUP and INF are more general however. Given a satisfiable set of constraints, an expression over the satisfying set, and a subspace, SUP and INF compute surfaces defined over the projection into the subspace of the satisfying set, which everywhere bound the expression over the inverse image of the projected points in the original satisfying set. More formally, but perhaps more clearly, suppose $support(\mathrm{C}) \subseteq W$, $V \subseteq W$, and e is an expression where $support(e) \subseteq W$. If

$$sat(\mathrm{C}, W) \neq \emptyset$$

then

$$support(\mathrm{SUP}(e, \mathrm{C}, V)) \subseteq V,$$

$$support(\mathrm{INF}(e, \mathrm{C}, V)) \subseteq V$$

and

$$\forall x \in sat(\mathrm{C}, W), \qquad (5.2.1)$$
$$[\mathrm{SUP}(e, \mathrm{C}, V)]_V(proj(W, V, x)) \geq [e]_W(x) \geq [\mathrm{INF}(e, \mathrm{C}, V)]_V(proj(W, V, x)).$$

Figure 10 gives an illustration of these capabilites. There the set W is defined by $W = \{x, y\}$, and $V = \{y\}$. The constraint set C is satisfied by the region shaded in the x–y plane. The expression e takes values in the reals $\Re$ over the satisfying set and gives rise to the surface patch illustrated. The darkened region of the y–axis illustrates the projection of the satisfying set $sat(\mathrm{C}, W)$ into $space(V)$. The shaded region in the y–$\Re$ plane is the corresponding projection of the values achieved by e. The curves in the y–$\Re$ plane above and below that shaded region correspond to the values achieved by the expressions returned by $\mathrm{SUP}(e, \mathrm{C}, V)$ and $\mathrm{INF}(e, \mathrm{C}, V)$, both expressions in y, over the porjection of the satisfying set. Notice that they are upper and lower bounds on the projection of the surface patch generated by e.

5.2.2 A partial decision procedure.

The procedures SUP and INF can be combined (following Bledsoe (1975)) to produce a partial decision procedure on sets of constraints. The decision concerns whether a set C of constraints is consistent, i.e. whether the constraints are satisfiable, or formally whether

$$sat(\mathrm{C}, support(\mathrm{C})) \neq \emptyset.$$

That the procedure is partial comes from the fact that it can not always decide whether or not this is the case. In fact it has two outcomes; one that the satisfying set is definitely empty, and the other that it doesn't know. This property is the cost of requiring that it always terminate in some bounded time determined by the size and complexity of the constraint set C.

The decision procedure is called DECIDE. Given a set of constraints C, then if

$$\forall v \in support(\mathrm{C}),\ \mathrm{INF}(v, \mathrm{C}, \emptyset) \leq \mathrm{SUP}(v, \mathrm{C}, \emptyset)$$

(where the definition of "$\leq$" is extended to handle $\pm\infty$ correctly) DECIDE(C) returns *"possibly satisfiable"* (or *true*), else it returns *"definitely unsatisfiable"* (or *false*).

The procedure DECIDE is sound in the sense that it never returns an incorrect result.

This follows from the fact that SUP and INF return upper and lower bounds over the satisfying set of a set of constraints.

Of course a procedure which always returns *"possibly satisfiable"* is also sound under this definition of soundness. Such a procedure happens to be worthless, however. A partial decision procedure is only interesting is it sometimes detects unsatisfiable sets of constraints. The more often it successfully detects such sets, the more interesting it is.

The only characterization of the extended SUP-INF decision procedure is empirical. In practice, in its use in the ACRONYM system there was never a case observed where it failed to detect an inconsistent set of constraints. However it is possible to construct a set of constraints which is in fact inconsistent, on which DECIDE returns *"possibly satisfiable"*. The philosphy adopted in ACRONYM was that if there was a failure to detect an inconsistency at some point in the computation, it would more than likely be detected later as the implications of the inconsistency were propagated and became less subtle.

5.3 A critical sub-procedure.

Besides DECIDE, another important sub-procedure used in the plan checker is PROJCS. It projects a set of constraints into a subspace over the satisfying set of a second set of constraints in such a way that points which satisfy the projected constraints also satisfy the original constraints. Essentially it tries to find prismatic subsets of the satisfying set of the first set of constraints, with elongation orthogonal to the projection subspace, which are wholly contained in the satisfying set of the second set of constraints.

Given variables sets W and V where $V \subseteq W$ and constraint sets C_1 and C_2 where $support(C_1) \subseteq W$ and $support(C_2) \subseteq W$, the procedure PROJCS simply computes the projection of C_1 from $space(W)$ to $space(V)$, over the satisfying set of C_2, by

$$\text{PROJCS}(W, V, C_1, C_2)$$
$$= \{\text{"SUP}(a, C_2, V) \leq \text{INF}(b, C_2, V)\text{"} \mid \text{"}a \leq b\text{"} \in C_1, \text{SUP}(a, C_2, \emptyset) \not\leq \text{INF}(b, C_2, \emptyset)\}.$$

If a constraint contains conjunctions or disjunctions then PROJCS simply maps this projection over the terms. Thus PROJCS is a computable procedure which returns a set of

constraints. The set V supports those constraints. The comparison of the numeric upper and lower bounds of a and b respectively simply serves to prune out constraints which are trivially true over the satisfying set of C_2. The key property of procedure PROJCS is given by the following lemma.

LEMMA: Let $C = \text{PROJCS}(W, V, C_1, C_2)$. Then

$$sat(C \cup C_2, W) \subseteq sat(C_1, W).$$

PROOF: Let $x \in sat(C \cup C_2, W)$, and let $a \leq b$ where a and b are expressions in W be a constraint in C_1 which is not trivially satisfied over $sat(C_2, W)$.

Since $x \in sat(C_2, W)$, then by the definition of function SUP (see (5.2.1))

$$[a]_W(x) \leq [\text{SUP}(a, C_2, V)]_V(proj(W, V, x))$$

and

$$[\text{INF}(b, C_2, V)]_V(proj(W, V, x)) \leq [b]_W(x).$$

But $x \in sat(C, W)$ also and hence satisfies every constraint in C. In particular C includes the constraint

$$\text{SUP}(a, C_2, V) \leq \text{INF}(b, C_2, V)$$

and hence

$$[a]_W(x) \leq [b]_W(x).$$

Thus $x \in sat(C_1, W)$.∎

The procedure PROJCS often results in a set of constraints whose satisfying set is expressed as a disjunction of sets satisfying subsets of the constraints. This is because it construct inequalities by putting expressions produced by INF, which includes "max" expressions, on the right of "$\leq$" symbols, and expressions produced by SUP, which include "min" expressions, are put on the left.

5.4 Checking a plan.

The plan checker must check a sequence of subplans at a particular level of abstraction. A procedure CHECK is defined later in this section which checks individual subplans, or plan steps. It must be used by a higher level plan checker which typically would be part of the actual robot planner. In this paper we assume a simple model for the plan checking aspect of the robot planner.

First it is given an initial state of the world and it propagates the effects of actions on the world, using CHECK, through the plan steps while constraining the plan variables so that all actions are guaranteed to succeed. Whenever a plan step is reached which can not be guaranteed to work by constraining plan variables or sensing at that step of the plan, the robot planner backs up, again using CHECK, carrying back a set of goal constraints to a point where CHECK can guarantee that the goal can be satisfied. It then proceeds forward again from that subplan aplying CHECK and propagating the results.

This process is illustrated below by checking the four plans A through B introduced in section 2.2.

Clearly there is room for much work on the role of the robot planner and the negotiations which can take place between adjacent planning islands. This paper has not directly addressed that issue, as it is more properly part of the planning process itself, rather than part of plan checking *per se*. What this paper has done is to give a mathematical framework within which those negotiations can be explored.

5.4.1 Simple checking.

Figure 11 defines the procedure CHECKSIMP. It is the part of the plan checker which tests whether a plan can be guaranteed to work simply by constraining the plan variables.

In its argument list P is the plan to be checked and P^* the plan which follows it in the

```
procedure CHECKSIMP(P, P*, FEEDFWD)
  begin
    if not DECIDE(|C_I ∪ C_A]_U(0_U))
    then outcome 6
    else
          begin
            C_N ← PROJCS(P ∪ U, P, C_A, C_I)
                        ∪ (if FEEDFWD
                              then ∅
                              else PROJCS(P ∪ U ∪ V, P, C*_I, C_I ∪ C_G));
            if DECIDE(C_I ∪ C_N)
            then
                  begin
                    if FEEDFWD then PROPAGATE(P*, C_I ∪ C_N ∪ C_G);
                    return outcome 1
                  end
            else return other
          end
  end;
```

Figure 11. Procedure to check whether a plan will work if its plan variables are sufficiently constrained.

chain of planning islands. A flag, FEEDFWD, says whether the initial states of the following plan $\mathcal{P}^*$ should be used as a goal condition (when the flag is false) or whether the initial states of the following plan should be derived from the current plan (when the flag is true). In that case a procedure PROPAGATE is called to update $\mathcal{P}$. Details of that procedure are not considered here.

THEOREM: If procedure CHECKSIMP produces outcome 1 then the plan

$$(g, P, U, V, \mathrm{C}_I \cup \mathrm{C}_N, \mathrm{C}_A, \mathrm{C}_G)$$

is sound.

PROOF: It suffices to show that conditions **(A)** and **(G)** hold.

Since C_N was computed by procedure PROJCS the lemma of section 5.3 says that

$$sat(\mathrm{C}_I \cup \mathrm{C}_N, P \cup U) \subseteq sat(\mathrm{C}_A, P \cup U)$$

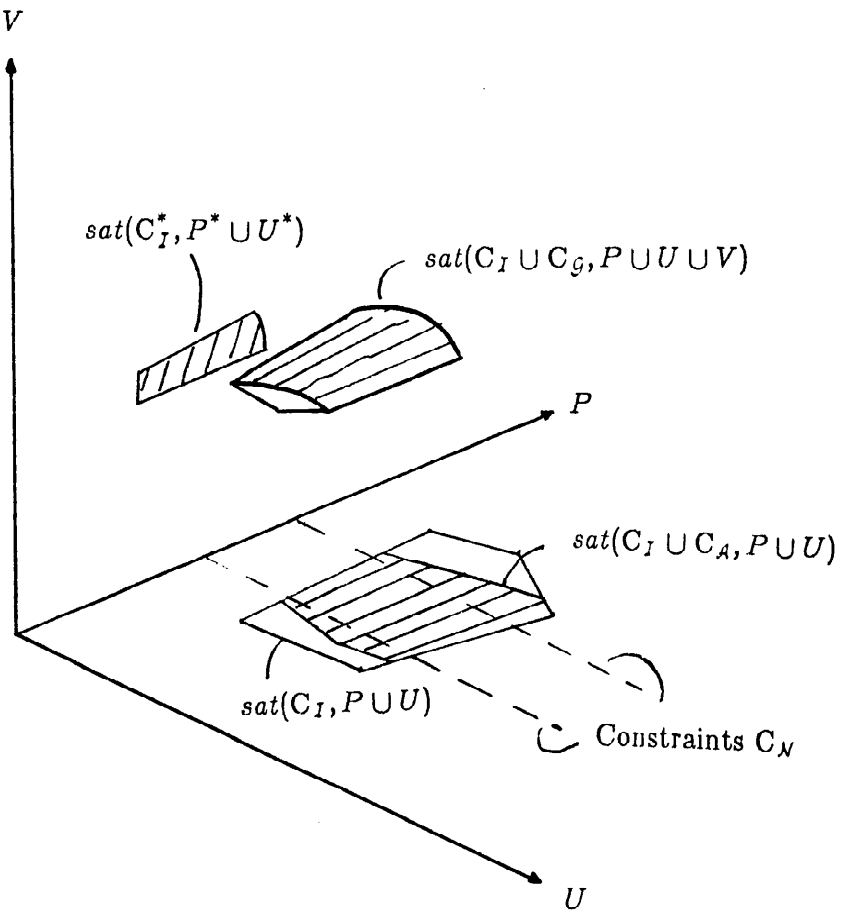

Figure 12. An illustration of the various constraint sets and their satisfying sets involved in finding sufficient constraints on plan variables to guarantee that a plan will succeed. The text contains detailed commentary.

whence condition **(A)** is proved.

If FEEDFWD was true then the theorem is proved. Otherwise the lemma can again be used to establish that

$$sat(C_I \cup C_G \cup C_N, P \cup U \cup V) \subseteq sat(C_I^*, P \cup U \cup V).$$

But $P^* \subseteq P$ and $U^* \subseteq U \cup V$ whence condition **(G)** is satisfied. ∎

5.4.2 A more intuitive explanation.

This section tries to give a more intuitive explanation of what is going on in constructing the constraints C_N above. Consider figure 12.

For simplicity (and drawabilty!) the sets P, U and V have been compressed to single variables. In addition the axes have been offset so that they don't go through the zero values of variables in U and V. The large region outlined in the P-U plane is $sat(C_I, P \cup U)$ — the set of initial states possible for the original plan. The region in the P-V plane is $sat(C_I^*, P^* \cup U^*)$. Note that in this case $P = P^*$ and $V = U^*$. The smaller region in the P-U plane is the region where the action of plan P is applicable - i.e. $sat(C_I \cup C_A, P \cup U)$.

The surface floating above is the set of states which the action can achieve when applied to an initial state. In this diagram there is only one resultant state per initial state, so that it could be represented by a function. The surface patch is given by $sat(C_I \cup C_G, P \cup U \cup V)$.

The role of procedure CHECKSIMP is to further restrtict the set of initial states to where the action is applicable and to where the resultant final state will project into the set of initial states of the following subplan P^*.

To avoid wishful thinking (see section 4.2) the cross section of the new set of initial states must be identical in the U direction wherever it intersects the original set. This is because one cannot change the initial uncertainties purely by legislation — sensing must be introduced if that is desired. Therefore the only constraints allowed in this diagram are ones on P, and so they must be parallel to the U-axis.

Condition (**A**) says that the initial states should be confined to be a subset of the points which satisfy C_A. Condition (**G**) says that the initial states should be confined so that the patch of final states above them projects into the initial set of the following subplan, in the P-V plane in this case. The dashed lines give constraints C_N which guarantee that conditions (**A**) and (**G**) are satisfied.

```
procedure CHECK(P, P*, FEEDFWD)
  begin
    case CHECKSIMP(P, P*, FEEDFWD)of
      1: return outcome 1;
      6: return outcome 6;
      other: begin
               C_N ← C_A ∪ (if FEEDFWD
                              then ∅
                              else PROJCS(P ∪ U ∪ V, P ∪ U, C*_I, C_I ∪ C_G));
               S ← SENSEVARS(C_I, C_N, P, U);
               P+ ← RESTRUCTURE(P, S);
               case CHECKSIMP(P+, P+*, .true.) of
                 1: return outcome 2;
                 6: return outcome 6;
                 other: if PROPBACK(P, C_N)
                          then return outcome 3
                          else return outcome 5;
                endcase
             end;
    endcase
  end;
```

Figure 13. The main plan checking algorithm.

5.4.3 Full scale checking.

Figure 13 gives the main plan checking algorithm. It uses CHECKSIMP to check the original plan and simply passes on the result if the plan either fails to work completely or if it can be guaranteed to work by simply constraining plan variables. Otherwise CHECK attempts to introduce sensing into the plan to see if that will help.

It constructs a new set C_N, using PROJCS as does CHECK, but this time its support can include uncertainty variables from U. It calls a procedure SENSEVARS described in section 5.5 below to decide which of the uncertainty variables need to be reduced by sensing to meet the new constraints C_N. SENSEVARS returns a set of physical quantities to be measured, and associated sensors to do the measurement.

The procedure RESTRUCTURE is invoked to carry out the operations described in

section 4.2 to restructure the plan $\mathcal{P}$ by introducing new plan variables whose values will be instantiated at plan execution time by interpreting the chosen sensors. (Section 5.5 below shows some practical simplifications which can be made to the constraints of section 4.2. The simplifications are not approximations, but rather conservative estimates which result in much simpler constraint sets and thus less computation time. Their drawback is that in very tight situations the plan checker may reject a plan which is actually valid.)

Now procedure CHECK invokes CHECKSIMP again to check the restructured plan. The FEEDFWD flag is passed true, so that subsequent planning islands will be restructured for the new sensing variables. If CHECKSIMP says that the restructured plan $\mathcal{P}^+$ is sound then CHECK is done and by the theorem above the plan with sensing is sound. Otherwise the procedure PROPBACK is invoked. It recursivley re-invokes the plan checker on the previous (and already checked) plan, with FEEDFWD false, to see if it is possible to deliver the world to plan $\mathcal{P}$ in a state where the uncertainties meet the constraints $C_{\mathcal{N}}$. Of course the result of CHECK on the previous plan may again result in PROPBACK being invoked, and further recrusive calls of CHECK back through the chain of plans. If that recursion finally fails the PROPBACK returns false and procedure CHECK gives up by signalling outcome 5. If PROPBACK is successful then $C_{\mathcal{N}}$ defines a new plan which is again sound and outcome 3 is the result. Examples of the behavior of PROBACK are given in section 5.6 below.

5.4.4 An example.

The function CHECK is called on each of plan A through plan D succesively, with FEEDFWD true, so that the initial constraints of the following plans are generated. Section 3.3 showed the initial states for plan B which had been generated from plan A. In that case the call to PROJCS in CHECKSIMP produces no new constraints, as the constraint from $C_{\mathcal{A}}$, namely

$$12.0 \leq \texttt{BOX-POS} \leq 36.0$$

is trivially satisified.

Plan B generates no new state information. Finally plan C generates initial constraints of

$$12.0 \leq \texttt{BOX-POS} \leq 36.0$$
$$e_l(\texttt{BOX-POS}) \leq \texttt{BOX-UNC} \leq e_h(\texttt{BOX-POS})$$
$$e_l(\texttt{BOX-POS}) \leq \texttt{LID-UNC} \leq e_h(\texttt{BOX-POS})$$
$$e_l(\texttt{BOX-POS}) \leq \texttt{BOLT-UNC} \leq e_h(\texttt{BOX-POS})$$

where

$$e_l(x) = \max(0.0002215x - 0.043262, 0.0009857x - 0.063329)$$
$$e_h(x) = \min(0.043262 - 0.0002253x, 0.063329 - 0.0009895x)$$

for plan D. Again PROJCS generated no new constraints in this case. Note that

$$P = \{\, \texttt{BOX-POS} \,\}$$
$$U = \{\, \texttt{BOX-UNC}, \texttt{LID-UNC}, \texttt{BOLT-UNC} \,\}.$$

When CHECK is applied to plan D the first call to CHECKSIMP fails with outcome "other". The reason, although the plan checker can not isolate it to this level of analysis, is that all the bolt and the lid line up well enough for initial insertion, there is too much uncertainty in the relative positions of the box and lid to guarantee that the holes in them line up. A new set C_N is computed where $support(C_N) \subseteq P \cup U$. The procedure SENSEVARS is invoked and it deduces that BOX-UNC and LID-UNC have smaller ranges of values when C_N constrains the intital states (SENSEVARS is described in more detail in the next section). It correctly deduces that there is no need to reduce BOLT-UNC anywhere in the range of box positions.

The plan is restructured to introduce sensing. However the procedure RESTRUCTURE notes that no named physical quantities are introduced in plan D, so sensing can not affect any action which is to take place. Therefore it immediately invokes the procedure PROPBACK. It propagates the set $C_I \cup C_N$ back to plan C as the initial constraints of plan D. Thus when CHECKSIMP is invoked on plan C with flag FEEDFWD set to false, it is realized that plan C needs to produce a more tightly constrained world state than before. Again SENSEVARS is invoked and it recommends reductions in the ranges of BOX-UNC and LID-UNC.

Procedure RESTRUCTURE is once again invoked. It decides that since there is only

one new physical quantity to be introduced, namely *bolt*:position, and since it depends only on *lid*:position, then it can only possibly help to sense the latter, and thus it can only possibly help to reduce the uncertainty in the position of the lid and not in the position of the box. Sensing is introduced, and then PROPBACK propagates forward from the new plan C to plan D once again. However plan D once again fails as the relative positions of the box and lid have still not changed, and so there is no guarantee that they will line up well enough for the bolt to be inserted through them both. Therefore PROPBACK continues to back up from plan C. The result is described below in section 5.6.

5.5 Introducing sensing.

Section 5.4 described procedures to check plans. A major subprocedure invoked by CHECK was SENSEVARS. Its job is to select physical quantities and sensors to be introduced into a plan to guarantee its success. There are two stages to that process. First the uncertainties which must be reduced by sensing need to be identified. These can be determined by examing the set of new constraints C_N. The prcoess is described below. The second step is to choose sensors which will indeed reduce those uncertainties.

5.5.1 Deciding what needs to be sensed.

Refer to figure 14. The outlined area in the P–U plan is the original set $sat(C_I, P \cup U)$. The shaded subset of that area corresponds to $sat(C_I \cup C_N, P \cup U)$. For a given value of the single plan variable in P a piece of the original set which is unshaded represents a tightening of constraints on the single uncertainty variable in U. In general it will be necessary to identify which uncertainty variables are so constrained.

Consider an uncertainty variable $u \in U$ and the problem of deciding if it has been further constrained. Let

$$o_s = \mathrm{SUP}(u, C_I, P \cup U - \{u\})$$
$$o_i = \mathrm{INF}(u, C_I, P \cup U - \{u\})$$
$$n_s = \mathrm{SUP}(u, C_I \cup C_N, P \cup U - \{u\})$$
$$n_i = \mathrm{INF}(u, C_I \cup C_N, P \cup U - \{u\}).$$

Expressions o_s and o_i are the original bounds on u, (expressed as functions of the parts of space orthogonal to u) and n_s and n_i are bounds on the newly constrained u.

If u is indeed constrained by the set C_N then there exists some point

$$x \in space(P \cup U - \{u\})$$

at which one of

$$[n_s]_{P \cup U - \{u\}}(x) < [o_s]_{P \cup U - \{u\}}(x) \quad (5.5.1)$$
$$[n_i]_{P \cup U - \{u\}}(x) > [o_i]_{P \cup U - \{u\}}(x) \quad (5.5.2)$$

is true. In the remainder of this analysis only o_s and n_s will be considered. Dual statements hold for n_i and o_i.

To determine whether condition (5.5.1) were ever true it would suffice to examine $o_s - n_s$ and see if it were ever positive. However o_s and n_s will typically be expressions involving "min" and thus their difference will be too complex for the SUP–INF method.

Suppose however that $n_s = \min(n_{s1}, n_{s2}, \ldots, n_{sk})$. Then if for any $\epsilon > 0$ one of the sets

$$C = C_I \cup \{u \geq n_{sj} + \epsilon\}$$

is satisfiable then condition (5.5.1) is satisifed.

Therefore the algorithm SENSEVARS simply chooses some small ϵ and for each u applies procedure DECIDE to each set of the form C above. If at least one of the sets is shown to be possibly satisifiable then u us a candidate for reduction. Note that if for some u all sets are said to unsatisfiable by DECIDE then (since DECIDE only says sets are unsatisifiable when indeed they are unsatisfiable) u has nowhere been reduced by more than ϵ over $space(P \cup U - \{u\})$.

In the example of sections 5.4.4 and 5.6 a constant ϵ of 0.000001 was used.

5.5.2 Choosing a sensor.

Once the uncertainties are to be reduced by sensing have been chosen, it is necessary to carry out some geometric reasoning to find which quantities can be measured in order to reduce those particular uncertainties. Each uncertainty is associated with a particular named physical quantity in the geometry g of the plan. A quantity which can be sensed and which depends on that named physical quantity must be found. That topic is not covered in this paper.

Once a candidate sensor has been found it would be advantageous to determine immediately if measurement error is small enough to provide the desired reduction in uncertainty. If more that one candidate sensor is found then this capability would be even more desirable. The SUP-INF method proves useful in this task also, but more work remains to be done on the topic and meanwhile a useable plan checker can be implemented based on the SUP-INF method without this capability.

5.5.3 Deriving the constraints.

Section 4.4.2 defined the constraints which can be inferred from adding a sense operation into a plan. Those constraints contain many variables and an equality. Such constraints slow down the SUP-INF method significantly, making the analysis of a sensing operation expensive. It turns out however that a set of simpler constraints, essentially projections of the exact forms into a subspace, are almost as strong as the originals. Thus it is possible to trade time spent in plan analysis for the possibility of missing a correct plan when all constraints are extremely tight. This exactly what the plan checker implemented by the author does.

In addition the implemented plan checker assumes that the robot controller will use the nominal value returned by a sensor as the new nominal value for the physical quantity being sensed. The validity of such sensor interpretations depends on the ability at plan checking time to be able to split up a satisfying set of C_I into components where the

physical quantity ranges over a single interval.

Recall the notation used. A physical quantity represented by the expression $o + v$, where o has support in P and v has support in U, is to be sensed and to be represented by the expression $n + u$ where n represents the nominal value and u the uncertainty. The implemented plan checker introduces the following three constraints.

$$\begin{aligned} n + l_s(n) &\leq o + \mathrm{SUP}(v, \mathrm{C}_I, support(o)) \\ o + \mathrm{INF}(v, \mathrm{C}_I, support(o)) &\leq n + r_s(n) \\ l_s(n) \leq \quad u \quad &\leq r_s(n) \end{aligned}$$

The first two are an expression of constraint (4.3.1) projected into a subspace of plan variables. The projection is conservative in the sense that anything that satisifies these two constraints will also satisfy (4.3.1). Thus the plan checker will have more situations to deal with than might have been described by using the more exact constraint. Since no explicit sense variable is introduced, but n is used directly instead, these two constraints also express the relation given by constraint (4.3.3). The final constraint concerns the uncertainty in the sensed quantity, and it takes the place of constraint (4.3.4).

Notice that the first two introduced constraints allow n, the new nominal value, to have a larger range of values than o, the older. Practically this often means that some constraint, newly expressed in terms of n, will cut down the allowed range of values for o so that n ends up having the same range as did o before the sensing constraints were introduced. It is this process which validates the use of the nominal sensor reading as the nominal value for the quantity being measured.

5.6 Completing the example.

Consider again the example of the four coupled plans.

After rejecting the introduction of sensing at the start of plan C the procedure

PROPBACK works back through plan B and plan A. At plan B it notices that no new physical quantities are introduced, so sensing can not help. The extra constraints requesting a reduction in the uncertainty of the box and lid positions are carried back to plan A.

Procedure SENSEVARS suggests that the box position should be sensed. Since an introduced physical quantity *lid*:position depends on *box*:position the procedure RESTRUCTURE introduces sensing for that quantity.

Now the new constraints are propagated through the series of plans using procedure CHECKSIMP. Extra constraints get added to the new plan A to ensure that the nominal value of the sensor can be used as the nominal value of *box*:position. For instance when the sensor has error characteristics

$$-l_s(m) = r_s(m) = 0.0004 \times m$$

then the constraint

$$12.0454 \leq \texttt{BOX-POS} \leq 35.9579$$

is added. No extra constraints C_N are needed until plan D is reached. The final subplan places extra constraints on the plan variables (the initial nominal position of the box, BOX-POS is the only plan variable), depending on the error characteristics of the sensor. The following table summarizes some results.

Function l_s	Function r_s	Resulting C_N
$-0.00035 \times x$	$0.00035 \times x$	*empty*
$-0.00040 \times x$	$0.00040 \times x$	BOX-POS $\leq 20.0892 \vee 28.1397 \leq$ BOX-POS
$-0.00045 \times x$	$0.00045 \times x$	BOX-POS $\leq 15.6811 \vee 30.7618 \leq$ BOX-POS
$-0.00050 \times x$	$0.00050 \times x$	BOX-POS $\leq 12.8564 \vee 33.9237 \leq$ BOX-POS
$-0.00055 \times x$	$0.00055 \times x$	*unsatisfiable*

The sensors, with linear error, smoothly degrade the range of initial box positions which lead to final success of plan D. With an error factor of 0.00035 the plan can be successfully

carried out wherever the box is initially placed. For 0.00040, 0.00045 and 0.00050 the central region of the working area of the manipulator is forbidden, as the combination of sensor error and manipulator uncertainty makes the plan infeasible. If the only sensor available has an error factor of 0.00055 then nowhere is sensing powerful enough.

In the latter case when the plans are propagated forward from plan A with sensing through to plan D, the plan checker finds that the set C_N computed by procedure CHECKSIMP leads to no feasible initial positions for the box. There is nowhere left to propagate backwards from plan A so it must conclude that in this case the sequence of plans is infeasible due to inadequate sensing.

6. Related Questions

This paper has developed a formal model for checking robot plans. To do so it first developed a formal model of plans. Plans are the result of planning. The plan checker is able to modify plans, but there has been no development in this paper of the formal process of planning. Thus there are some deficiencies in the model of plans used. Work is needed to integrate the model presented here with more creative planning processes.

6.1 Planning.

Most work on planning has been restricted to abstract domains where there is little notion of a metric, let alone spatial relations, errors or tolerances.

Sussman's (1975) HACKER program works entirely in an abstract blocks world. Sacerdoti's (1977) NOAH is not restricted to the blocks world, but every problem description requires a procedural semantics of the domain to accompany it. The user really needs to know the solution in advance in order to decide the form of the semantic description. Stefik's (1981) MOLGEN works in a domain of planning molecular genetics experiments, but it doesn't even have a strong notion of quantity. All these programs concentrate on planning, but in a domain with impoverished semantics. The semantics of their worlds can be completely specified in a few lines. They plan within that very simple world. Many of their ideas are useful for robot planning, but one should not suppose that any of these programs are capable of producing plans that could possibly work in a real world.

The ABSTRIPS sytstem (Sacerdoti (1974)) is often cited as a real world robot planning system. Indeed, a physical robot, SHAKEY, was controlled by plans generated by ABSTRIPS. However, SHAKEY was restricted to a tightly controlled environment, and much careful engineering ensured that the world was relatively benign. Errors that the real world introduced were handled by having the complete planing system available at execution time (in PLANEX) to modify the pre-computed plans. As such, this is an excellent idea, where runtime processing power makes it viable, to handle deviations in the modelled

world from that encountered in the real world. It has the disadvantage of the execution time strategy turning into somewhat of a "hit-and-miss" affair, where the models are not completely adequate, so a cycle of "action, sense, new action to achieve the same goal", can often develop.

Fahlman's (1973) BUILD system, incorporated three dimensional models of objects from the blocks world. Uncertainties of size and position ((1973) p. 48) were ignored as an extra complication to be tackled later. Fahlman did however deal with objects touching, gravity and frictional forces. BUILD is capable of impressive behavior in the face of complex arrangements of blocks where these forces are significant. Fahlman claimed that "80% of the performance" of his system derived from the level of detail in the model it was given of a particular situation, while the remainder came from the planning knowledge embedded in the system. Further, he claimed that the planning aspects are greatly simplified by having the detailed models. His system is probably unable to deal with real-world problems, but it is much closer than the other planning systems mentioned above.

Moravec (1980) programmed a mobile robot to navigate through a cluttered environment using a visual map. It took into account errors and their effects at many stages of its computations. Because of its simple model of the world as clusters of points with known three space coordinates it had to be very generous in enclosing them with spheres to be avoided. It had a self model of how commands to its motors would affect its three dimensional location and orientation. This model took into account frictional forces, and errors, but always assumed worst case errors to analyze the possible outcomes. Features that were located by its nine eyes were also considered to be prone to error. All of Moravec's computations were of the form of propagating errors forward, then checking the result.

Taylor (1976) has carried out by far the most realistic analysis of errors in robot planning to date. He used some symbolic geometric reasoning techniques, and many powerful numerical techniques. The example in section 2.1 of this paper is originally due to Taylor. His work differs from that here, in that he essentially propagated errors and tolerances numerically forward through a physical situation, then checked at the end

whether such things as applicability conditions were met. If not, the plan was either rejected or modified at the point where the applicabilty condition was violated (e.g. initiate a spiral search using force sensing to find the hole for the screw to fit in).

The methods presented in this paper have their roots in Taylor's work. Instead of numerical computation in one direction, the computation (of the constraints) is symbolic; the computations can be proceed in any chosen direction.

Ambler and Popplestone (1975) and Popplestone, Ambler and Bellos (1980) have developed an assembly programming system which has many elements of planning in it. They use geometric models of objects and infer spatial locations and orientations of parts from relationships between them (e.g. "against" or "fits"). They do not take into account tolerances, but instead work with nominal representations of distances and angles. Within the framework of this paper much of what they do can be characterized as finding constraints on plan variables.

6.2 Integration of planning and checking.

The details of how the implemented plan checker propagates information from plan island to plan island, and exactly how it might fit in with an automatic planning system were to a great extent ignored in this paper. That is largely because there are many ways to integrate such plan checkers with planning systems.

A plan checker such as presented here could be integrated with a robot programming language at a number of different levels. The programmer might be required to model explictly the uncertainty effects of each individual motion command, along with providing explicit pre-conditions on the applicability of that motion. The plan checker could then treat every motion command as an individual plan island in a sequence of planned steps, check its validity and propagate the uncertainties to the next motion command. Like a smart compiler it could provide error messages about why the sequence of motions might be invalid. In a higher level robot programming language which included geometric models

of the manipulator and objects in the world, applicability and geometric constraints could be inferred more automatically by examing the motion statements. Again, advice could be given to the human programmer concerning critical points in the program where sensing was needed, or new motion commands were required.

A more automatic planning system culd also benefit from the plan checking approach presented in this paper. Both Lozano-Pérez (1976) and Taylor (1976) have discussed planning systems which expand skeleton program fragments into complete robot programs by taking into account the constraints implied by the geometry of the particular world instance. Often there are decisions to be made in fleshing out such skeletons which must rely on global interactions. The plan checker presented here could be used as such a decision tool. At the same time it can be used as the "test" part of a "generate and test" approach which the planner might use for decisions which its expertise gives no guidance.

The RAPT system of Popplestone, Ambler and Bellos (1980) takes over some of the planning burden from the human programmer. From the discussion of the previous section it is clear that much of what the plan checker does is in the spirit of the RAPT system's approach to planning by solving relational constraints. It seems that a plan checker following the framework described in this paper could be naturally built on top of RAPT.

6.3 The form of constraints.

The constraints used in the implemented plan checker are all algebraic inequalities. Ideally one would like to find ways to express much more geometric constraints and develop partial decision procedures that could deal with them in a manner adequate for using the model of plans presented in this paper. The algebraic constraints used to date, constrain parameterizations of, essentially, "topologically" equivalent situations. They give no hint how to talk about classes of states whose members include radically different topologies.

The model of plans (but not of sensors) given in section 3 is independent of the variables being real-valued or the constraints being inequalities. This gives some hope that it may be

possible to extend the plan checking formalism into the area of geometric checking, besides the the purely algebraic aspects presented here.

6.4 Summary.

This paper has concentrated on the algebraic aspects of robot plans that include explicit models of uncertainties in the physical world. Earlier work (Brooks (1981a)) has shown how to map from the geometry of a world model to the algebraic aspects studied here. A formal model of plans was developed, and a formal model of a plan checker followed. The details of implementing a particular plan checker were discussed, and it was shown that the plan checker could check the plan, constrain certain decisions and introduce sensing in order to ensure that the plan would work.

Acknowledgements

The presentation of the work in this paper has benefited materially from careful readings of drafts by J. Michael Brady and Tomás Lozano-Pérez.

References

Albus, J. S. and Evans, J. M. 1976. *Robot Sytems,* Scientific American, Feb., pp. 76-86B.

Ambler, A. P. and Popplestone, R. J. 1975. *Inferring the Positions of Bodies from Specified Spatial Relationships,* Artificial Intelligence (6):175–208.

Bledsoe, W. W. 1975. *A New Method for Proving Certain Presburger Formulas,* IJCAI-75, Tibilsi, Georgia, U.S.S.R., Sept., pp. 15–21.

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

Brooks, R. A. 1981b. *Symbolic Reasoning Among 3-D Models and 2-D Images,* Ph.D. dissertation, Stanford AIM–343, June.

Drake, S. 1977. *Using Compliance in Lieu of Sensory Feedback for Automatic Assembly,* Charles Stark Draper Lab. Report T–657, Sept.

Fahlman, S. E. 1973. *A Planning System for Robot Construction Tasks,* M.S. dissertation, MIT AI TR–283, May.

Lozano-Pérez, T. 1976. *The Design of a Mechanical Assembly System,* M.S. dissertation, MIT, AI–TR–397, Dec.

Minsky, M. 1963. *Steps Toward Artificial Intelligence,* in Computers and Thought, J. Feldman and E. A. Feigenbaum (eds.), McGraw–Hill.

Moravec, H. P. 1980. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover,* Ph.D. dissertation, Stanford AIM–340, Sept.

Popplestone, R. J., Ambler, A. P. and Bellos, I. M. 1980. *An Interpreter for a Language for Describing Assemblies,* Artificial Intelligence (14):79–107.

Sacerdoti, E. D. 1974. *Planning in a Hierarchy of Abstraction Spaces,* Artificial Intelligence (5):115–135.

Sacerdoti, E. D. 1977. *A Structure for Plans and Behavior,* American Elsevier.

Salisbury, J. K. 1980. *Active Stiffness Control of a Manipulator in Cartesian Coordinates* 19th IEEE Conference on Decision and Control, Albuquerque NM, Dec.

Shostak, R. E. 1977. *On the SUP–INF Method for Proving Presburger Formulas,* JACM (24):529–543.

Stefik, M. 1981. *Planning With Constraints,* Ph. D. dissertation, Stanford, HPP–80–2, Jan.

Sussman, G. J. 1975. *A Computer Model of Skill Aquisition,* American Elsevier.

Taylor, R. H. 1976. *A Synthesis of Manipulator Control Programs from Task–Level Specifications,* Ph.D. dissertation, Stanford AIM–282, July.

Appendix

The example introduced in section 2.2 and referred to throughout the paper has been checked by an implemented plan checker. The plan checker runs on a Lisp Machine at the Artificial Intelligence Laboratory at the Massachusetts Institute of Technology. Below is given the input specification to the checker which describes the four coupled plans. This was generated by hand, but in the future may be generated as the product of a task level planning system.

First the plan data-base is initialized and then the named physical quantities are declared. The entries POS and UNC simply declare the postfixes to be used in generating internal variable names, so that internal data structures can more easily be debugged. The POSITION entry defines the *type* of each named physical quantities (another example might be ANGLE).

```
;;; example of four coupled plans

(INITIALIZE-PLAN-DATA-BASE)

;;; named physical quantitites

(NPQDEC BOX POSITION POS UNC)
(NPQDEC LID POSITION POS UNC)
(NPQDEC BOLT POSITION POS UNC)
```

Functions are then defined to specify the uncertainty behavior of the manipulator. There is no specific model of the manipulator used by the plan checker at this level. The manipulator behavior is encoded in the sets of constraints C_g.

```
;;; upper and lower bounds on manipulator uncertainty

(FUNDEF EL (X)
  (MAX (+ (* 0.0002216 X) -0.043262) (+ (* 0.0009857 X) -0.063329)))

(FUNDEF EH (X)
  (MIN (+ (* -0.0002253 X) 0.043262) (+ (* -0.0009895 X) 0.063329)))
```

The error characteristics of the sensor are defined, and then a model of the sensor itself

is given. The model is in terms of its error characteristics and the *type* of named physical quantities which it can measure.

```
;;; sensor and bounds on its errors.

(FUNDEF LS (X) (* -0.0004 X))
(FUNDEF RS (X) (* 0.0004 X))

(DEFSENSOR CAMERA LS RS '(POSITION))
```

Now the four plans are defined. There are up to four slots defined for each plan. NPQ-INIT is a list of named physical quantities which describe the state of the world at entry to the plan. NPQ-ADDED-DEFS is a list of pairs describing named physical quantities introduced into the world state by the action of the plan. The first element of each pair is the name of the new quantity and the second is an expression whose nominal value will be used as a nominal value for the introduced quantity. Thus in plan A the lid is to be placed in the same nominal place as the box. The slots ABSTRACT-CA and ABSTRACT-CG describe the constraint sets $C_{\mathcal{A}}$ and $C_{\mathcal{G}}$ in terms of the named physical quantities. During the plan checking process they will be expanded in terms of plan and uncertainty variables. As sensing operations get introduced, those expansions will change.

```
;;; plan definitions

(DEFPLAN PLANA
         NPQ-INIT '(BOX-POSITION)
         NPQ-ADDED-DEFS '((LID-POSITION BOX-POSITION))
         ABSTRACT-CA '((IN (NOMINAL LID-POSITION) 12.0 36.0))
         ABSTRACT-CG '((IN (UNCERTAINTY LID-POSITION)
                           (EL (NOMINAL LID-POSITION))
                           (EH (NOMINAL LID-POSITION)))))


(DEFPLAN PLANB
         NPQ-INIT '(BOX-POSITION LID-POSITION)
         ABSTRACT-CA '((IN (- BOX-POSITION LID-POSITION) -1.0 1.0)))
```

```
(DEFPLAN PLANC
        NPQ-INIT '(BOX-POSITION LID-POSITION)
        NPQ-ADDED-DEFS '((BOLT-POSITION LID-POSITION))
        ABSTRACT-CA '((IN (NOMINAL BOLT-POSITION) 12.0 36.0))
        ABSTRACT-CG '((IN (UNCERTAINTY BOLT-POSITION)
                          (EL (NOMINAL BOLT-POSITION))
                          (EH (NOMINAL BOLT-POSITION)))))


(DEFPLAN PLAND
        NPQ-INIT '(BOX-POSITION LID-POSITION BOLT-POSITION)
        ABSTRACT-CA '((IN (- BOLT-POSITION LID-POSITION)
                          (% -7. 64.)
                          (% 7. 64.))
                      (IN (- LID-POSITION BOX-POSITION)
                          (% -3. 64.)
                          (% 3. 64.))))
```

Finally, the initial state of the world is specified, and plan *A* is appropriately initialized. The slot NPQ-LIST is filled with a list of named physical quantities corresponding to physical realities in the world's initial state, while ABSTRACT-CS is a list of constraints which describe the possible initial states of the world.

```
;;; initial state of the world

(INITIALIZE-PLAN PLANA
                 NPQ-LIST '(BOX-POSITION)
                 ABSTRACT-CS '((IN (NOMINAL BOX-POSITION) 12.0 36.0)
                               (IN (UNCERTAINTY BOX-POSITION)
                                   (EL (NOMINAL BOX-POSITION))
                                   (EH (NOMINAL BOX-POSITION)))))
```