DECISION ANALYSIS FOR AN EXPERIMENTAL ROBOT WITH UNRELIABLE SENSORS

L. Stephen Coles, Alan M. Robb, Paul L. Sinclair, Michael H. Smith, and Ralph R. Sobek
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720 U.S.A.

Abstract

This paper reports on the continuing design of and experimentation with Jason, the Berkeley Robot. Progress has been made in various aspects of the hardware (including the chassis, communications controller, and onboard microprocessor) and software (including problem-solving programs and world models). A particular experiment, analogous to the classical "Monkey and Bananas" Problem, is described. A major feature of the reformulation of this problem is the use of Decision Analysis in coping wJth uncertainty. Based on the accimulated expected costs of executing the steps of various hypothetical plans, Jason can evaluate the relative merits *of* direct *act ion* versus prior information-gathering using potentially unreliable sensors.

Introduction

As reported at the last IJCAI Conference,[1] the objective of our current research at the University of California at Berkeley is the design and implementation of a relatively inexpensive, general-purpose, computer-controlled, mobile robot. The Berkeley robot, dubbed Jason, was initially designed during the Spring of 1973 and tested off-line the following Summer. Hard-wire, blind-mode computer control was successfully *ac*-complishcd during January 1974. Unreliable breadboard wiring as well as greater than expected demands for on-board power have led to a substantial redesign effort; testing of a considerably improved version of Jason is now underway.

Our ultimate goal, as stated earlier, remains the investigation of the class of problems that a robot both encounters and creates while performing elementary tasks In a real-world environment ineluding active human beings. The results of this research will hopefully enable us to construct better and safer robots at a modest price that are still capable of performing a variety of useful tasks such as factory or ware-house work or can function as a teaching aid for young children in the classroom.

This paper is divided into five sections. The first briefly reviews some of the recent improvements to Jason hardware and software. The second section suggests some of the ways that robotics research differs from conventional Al problem-solving research. The third section presents a decision analysis formulation of the classical "monkey and bananas" problem. The fourth section provides the results of the decision analysis. Finally, the last section summarizes and proposes some improvements to Jason hardware and software for the future.

Jason Hardware/Software Improvements

Figures I and 2a-f provide an overview of the current state of Jason.

A. Hardware

First we discuss improvements to the key hardware subsystems: the chassis, the sensors, and the communications controller.

1. Robot Chassis--The Jason chassis consists of a half-inch thick aluminum base plate with an area of about two square feet with two additional aluminum shelves mounted above for electronic equipment- The three-ball configuration originally designed to support the front of Jason, although conceptually good, did not perform well on rough surfaces 1ike a parking lot, and consequently was replaced by a heavy-duty, 3-inch diameter, swivel-castor wheel. To provide additional on-board power, the conventional, 83-amp-hour, lead storage, auto battery was replaced by a 250-amp-hour, 150-pound, train battery. Due to extensive two-handed coordination problems, the original plan for two simple hands has been reduced to one, and a prosthetic arm with jaw gripper has now been installed. A simple two-posi tion-sensor push bar has also been installed: one setting indicates slight pressure contact while the other indicates too much pressure, i.e., Jason is trying to push a nonpushable object. The motor-control unit was also rebuilt to make it more rugged.

2. Sensors--An A-to-D converter to extract texture information as well as range data from the analog output of the ultrasonic torch has been dcsigned. An LED proximity sensor has been mounted on the arm, and the proximity-sensor interface was completed.

3. Communications Controller--The preliminary Jason 8-bit character asynchronous communications controller was successfully bench tested, re fabricated on circuit cards, and mounted in a new card rack and chassis by one of the authors (Robb) Previous problems of vibration should now be minimized, and circuit debugging should be greatly facilitated. Because the controller has been designed to be teletype-compatible, Jason could in principle be interfaced to any computer without special-purpose hardware being installed at the computer side. For example, Jason has already been connected to an HP-3000, a CDC-6400, and a PDP-10 computer system. Furthermore, we have recently demonstrated Jason over the ARPA Net. Further details on the controller can be found in Ref 2. Our FCC radio-license was renewed, and two-way radio telemetry has been successfully bench tested,

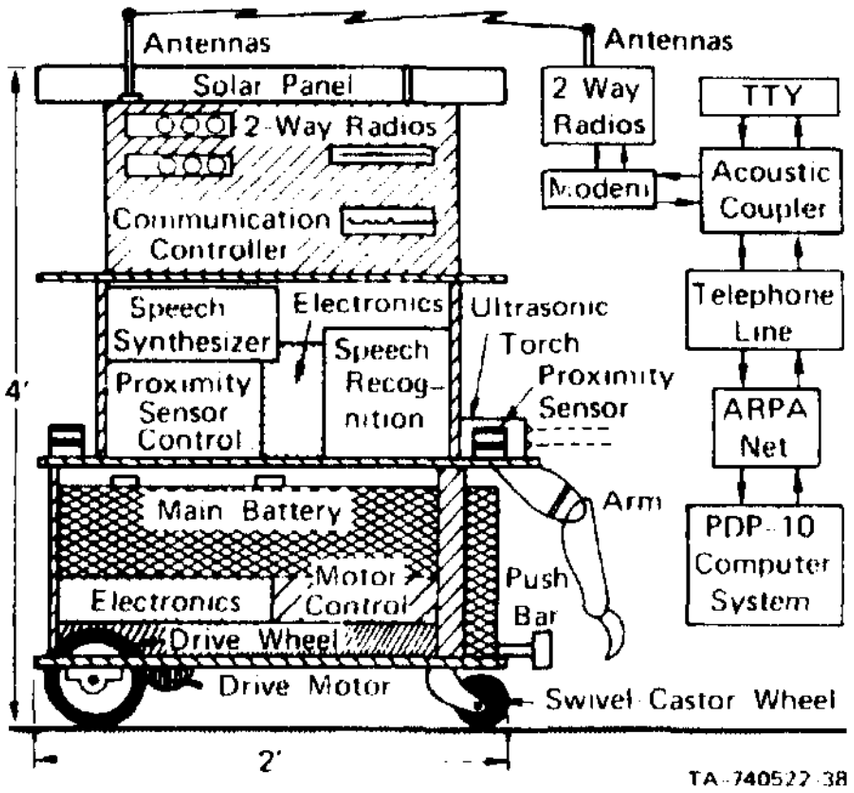


FIGURE 1 JASON, THE BERKELEY ROBOT

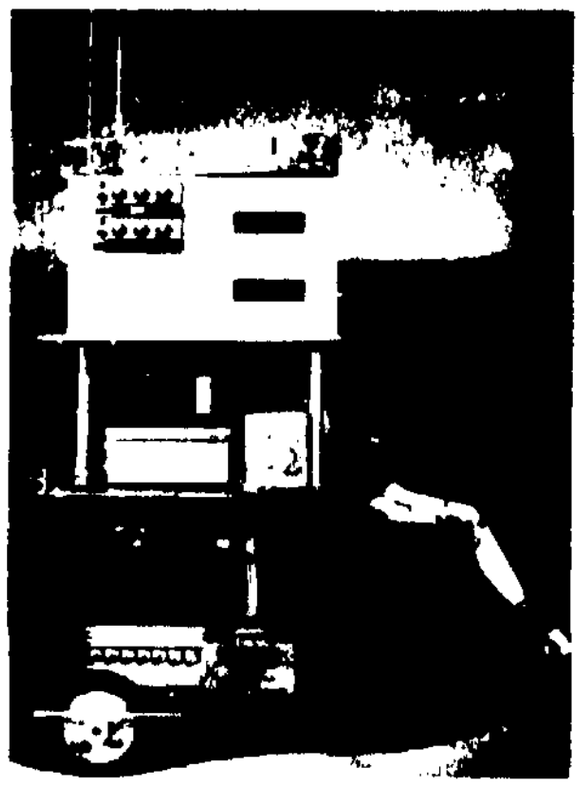


FIGURE 2a JASON-SIDE VIEW

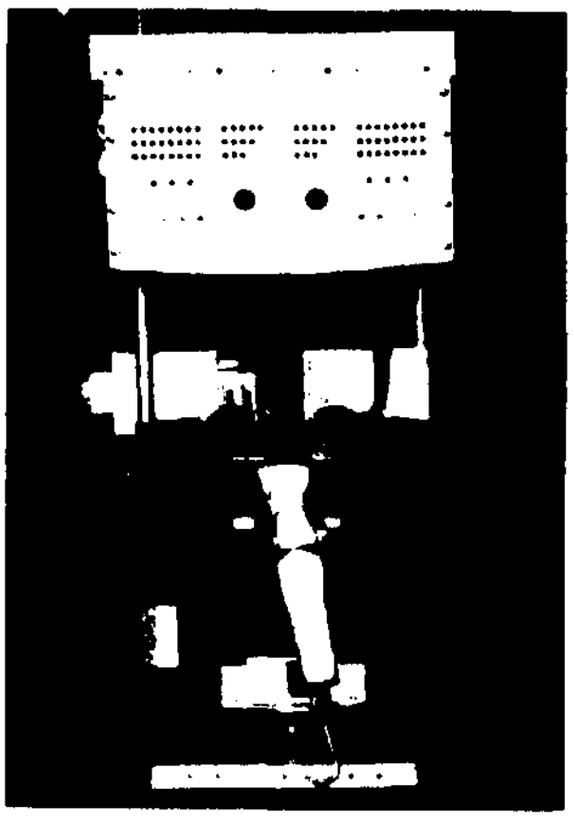


FIGURE 2b JASON-FRONT VIEW

FIGURE 2c JASON-REAR VIEW

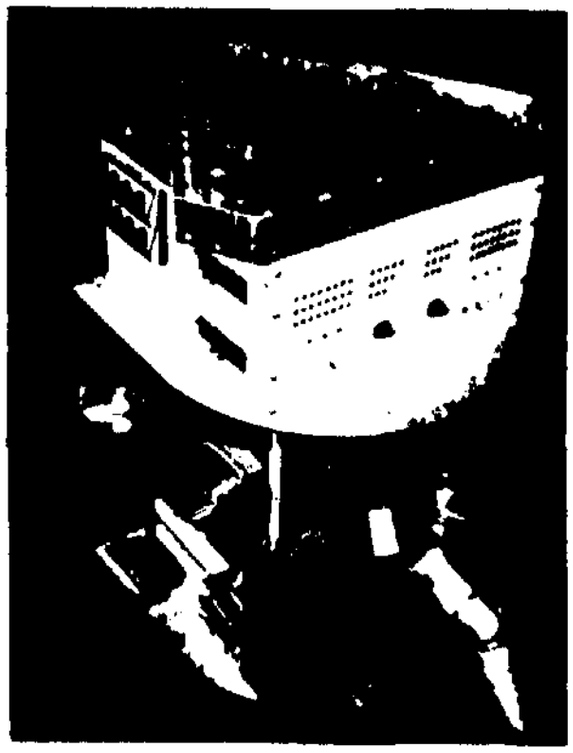


FIGURE 2d JASON-PERSPECTIVE VIEW

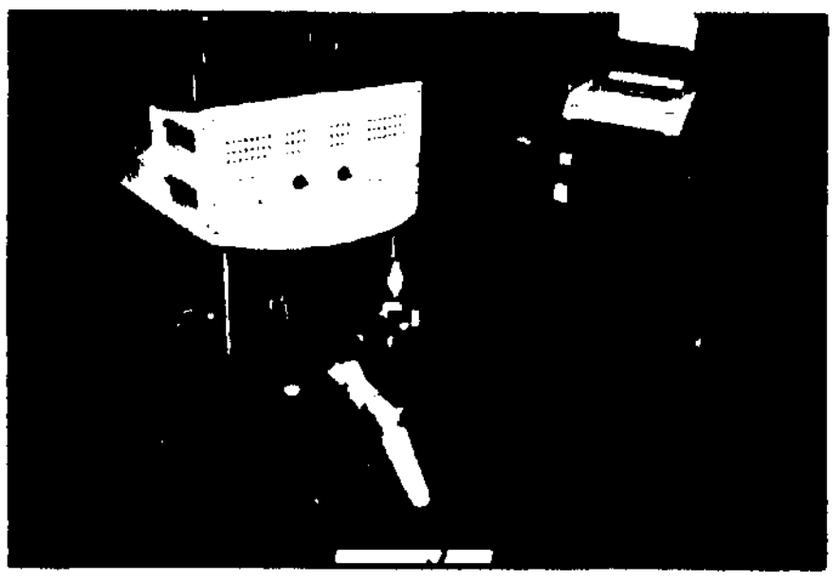


FIGURE 2e JASON WITH CONTROL TELETYPE

B. Software

On the software side, the check out of real-time interactive routines was handicapped by the removal of the HP-3000 computer from the Lawrence Hall of Science. Thus, all interactive work with Jason during the last year was simulated on the CDC-6400 available on the Berkeley campus. We have recently transferred Jason software to an interactive DEC PDP-10 computer system and have successfully tested Jason over the ARPA Net.

In the mean-time the Jason simulator, which has produced output tapes capable of actually driving Jason in blind mode, has grown extensively to include a much broader collection of Intermediate Level Operators (ILO's) such as Move, Turn, Face, Goto, Push, Pushto, Gotodoor, Gothrudoor, etc., capable of execution in an arbitrary collection of rooms connected by doorways and corridors, navigating optimally through an arbitrarily complex collection of boxes in any room. This work is documented in the Jason Reference Manual,[3] and although it represents the work of many individuals, two of the authors (Sinclair and Sobek) are largely responsible. Output from the simulator will be seen in Section IV.

Real-World Vs. Toy Problems

Early research in artificial intelligence tended to focus on a few well-known puzzles or "toy" problems, such as the Monkey and Bananas, Missionaries and Cannibals, or Tower of Hanoi problems (see Ref. 4 for these and other examples). Although toy problems demonstrate the intellectual competence of computers along one narrow dimension, research in robotics has broadened considerably the scope of the problem-solving enterprise. Our experience with the Jason project permits us to identify at least four general criteria which distinguish real-world robot problems from toy problems: problem-solving environment, problem formulation, data requirements, and solution requirements.

A. Problem-Solving Environment

A rough spectrum of problem-solving environments in terms of increasing complexity is indicated in the following table:

In the language of Table 1, almost all work in AI was conducted in a sterile environment (E1). Subsequent early robotics research began with surgically-clean environments (E2) and was severely criticized for giving the appearance of E3 to uneducated observers, when in reality it was not very much of an advance over E1. As more advanced scene analysis and perceptual techniques became available, emphasis shifted to E3 and E4. With the appearance of more sophisticated hardware sensors initial forays are being made into benevolent environments (E5). All work to date in Natural or Hostile Environments (E6,7) has been done with teleoperators, where a human is an essential part of the loop.

B. Problem-Formulation

Because of the real-time execution and training aspect of robotics work, the problem formulation must be accomplished in human terms rather than mathematically. Thus, voice-input, restricted natural-language-problem statements are desired. Other techniques to facilitate man-machine interaction such as "joy-sticks," cursor tracking balls, or light pens are also needed.

C. Data Requirements

Information to solve the problem may either be inadequate or embedded in a large quantity of seductive but mostly superfluous data, or some combination of these. Insufficient information might occur for a variety of reasons: (1) it may not be knowable in principle, (ii) it may not be known whether it is knowable in principle, (iii) the necessary information is knowable, but the cost of acquiring it may be prohibitive. (As a special case, the cost may be reasonable, but acquisition cannot be accomplished within sufficient time to be useful), and (iv) it is not known whether the cost of information-gathering will prove to be prohibitive. (In this case decision analysis may be useful.)

Once obtained, data may still possess uncertainty or lack credibility for various reasons: (1) the data may be incorrect due to statistical unreliability in the sensory path, (ii) the source is known to be prejudiced, (iii) the source is

	Environment	Description
E1.	Sterile	Mathematically precise, no interaction with the real world.
E2.	Surgically Clean	Real-world inputs, but highly contrived, such as painting objects with saturated colors so that the robot can identify them more easily.
E3.	Passive	Typical interior environment (such as an office) without people. Also, no wind or shadows due to changing illumination.
E4.	Active	Typical exterior environment (trees, sky, etc.), but without people or animals.
E5.	Benevolent	Active environment with cooperative humans exhibiting ruled behavior (such as in a factory or parking lot).
E6.	Natural	Benevolent environment including animals as well as seasonal weather changes.
E7.	Hostile	No constraints: mines, deep under water, outer space, etc., including potential adversaries.

TABLE 1 PROBLEM-SOLVING ENVIRONMENTS

known to be antagonistic and may deliberately provide false leads. Finally, (iv) the data may be inconclusive because of inherent ambiguities within the model upon which it is based and may not yield a definitive interpretation. Medical or meteorological data fall in this category.

D. Solution Requirements

As distinguished from conventional AI problem solving, a solution is not a solution for a robot until It has been successfully implemented as action in the real world. There are many opportunities for failure along the path to a solution in this sense: planning failures, execution failures, and decision failures (a failure of communication between the planner and the executor).

1. Planner Failures--Within the planner various failure modes may occur: Either a plan is found or it isn't. If no plan is found, it could either be due to the fact that none exists (i.e., the task is truly impossible) or the system lacked the intelligence to find one. If a plan is found, it might still fail for a variety of reasons: (i) it never could have worked (i.e., it fails in principle); (ii) it sometimes works (i.e., it fails in practice). A plan is said to be incomplete[6] (or soft) if it deliberately avoids dealing with all logically possible contingencies- Although such fragmentary plans are generally undesirable, they are sometimes preferable to no plan at all. Yet even a complete (or robust) plan will occasionally fail at some point in its capacity to sustain variation in environmental boundary conditions. If one were to plot the performance of a robot plan as a function of environmental complexity (El to E7 of Table 1) then one could characterize the failure mode as precipitous or capable of graceful degradation in proportion as whether the shape of performance fell off sharply or smoothly with complexity; (iii) even though questions of optimality are rarely stated explicitly in the problem formulation because it is normally unimportant whether the absolutely best way of doing something is proposed, plans should be penalized as failures if they are ludicrously inefficient in accomplishing the job. A plan post-processor might be useful in overcoming this kind of failure; (iv) the use of nondeterminIstic plans with parallel subsolution paths may make a plan more robust at the possible expense of introducing other failure modes.

2. Execution Failures--Execution failures come in two broad categories: internal and external. Internal failures sometimes called crashes, can either be hard or soft. A hard crash is not immediately recoverable, and the plan must be reinitialized. It may have been the result of either unreliable hardware or software. By comparison, a soft crash, usually due to a high-level monitor failure, will allow the robot to resume the plan where it left off, after a delay for reloading a fresh copy of the monitor. External failures are of three main types: (i) The robot failed, and knows that it failed. This may be due to either systematic or random errors in its operators when executed in the real world, since all actions have inherent uncertainty in their outcomes. (Better calibration should hopefully minimize systematic errors.) There may also be other legal error modes for operators, such as "timeouts" or resource-exceeded constraints; (ii) the robot failed, and didn't know that it failed (sometimes called a mlssense error). This may be due to infinite looping or other errors in the flow-of-control or to self-deception through imperfect sensing of the true state of affairs; (iii) the robot succeeded, but thinks that it failed (sometimes called a nonesense error). Illegal error messages or false alarms cause this kind of failure.

3. Decision Failures--Decision failures are much more subtle. Five general types will be mentioned: (i) too little or too much time devoted to planning compared with execution. This depends on the amount of time spent planning, the cost of thinking, the utility of the goal, the penalty for incurring undesirable irreversible state changes in the real world, and so forth. Note that human intuition may be very poor in this regard, since most human planning appears to take place at the subconscious level and therefore creates the illusion of being cost-free. Also, human planning and execution can frequently take place simultaneously, assuming that the execution process is not too intellectually-demanding. A robot may not always have this luxury; (ii) the failure to capitalize on serendipity. Tunnel vision during execution may cause the robot to push the solution out of the way in order to recreate the solution according to plan; (iii) failure to adequately reparameterize plans based on past experience. This is sometimes referred to as structural as distinguished from statistical learning; (iv) failure to reorder priorities dynamically. Multiple, possibly conflicting goals must be continually monitored during execution. This may lead to *seemingly* anomolous behavior from the point of view of an outside observer, but be perfectly consistent with internal objectives; (v) failure to distinguish local from global failures, i.e., calling upon the planner to replan from scratch, when salvaging the existing plan with a minor elaboration of an existing contingency branch of the current plan would be adequate, or conversely, trying al1 variations of a plan that was doomed to fail. In psychiatry this kind of pathology is referred to as functional fixity.

A New Formulation of the "Monkey and Bananas" Problem

The original formulation of the "Monkey and Bananas" problem[6] can be stated briefly as follows:

> A monkey is in a room in which a bunch of bananas are hanging from the ceiling, just out of reach. The monkey's problem, obviously, is to get the bananas. In the corner of the room is a chair. The solution

decided on by the monkey is to push the chair to a location under the bananas, climb on top of the chair, and then easily reach for the bananas.

The major interest of AI researchers in this problem is that it is characterized by one level of indirectness. That is, the solution requires an auxiliary device or tool (a chair in this case) not obviously needed at the start of the problem. In 1970 one of the authors (Coles) succeeded in formulating a fairly straight-forward transliteration of the problem into the world of the SRI robot.[7] The role of the tool was played by a ramp that allowed Shakey to push a box off a platform onto the floor, which he would not otherwise have been able to do. Others, such as McDermott,[8] have sought to generalize the information-gathering aspects of the problem.

In a more recent paper by Feldman and Sproull[9] this same problem has served as the basis for a decision-theoretic approach. Although they are much more concerned with the use of a numerical utility function during planning to develop a more efficient search strategy, we have independently reached the same conclusion regarding the positive value of joininR decision analysis with a symbolic robot problem solver to facilitate intelligent decision making under conditions of uncertainty. (See references 10-12 for an introduceion to decision analysis.) In particular, using this approach we can create a decision tree that allows one to 'Voll back" the consequences of further information-gathering operations in comparison with direct action. Of course, decision analysis can never guarantee a desirable outcome; it can on!y provide reasons why one course of action will in general be better than another.

Because of the inherent unreliability of Jason's ultrasonic texture/range-finder, we sought to find *a* realistic reformulation of the monkey and bananas problem that would also illustrate the value of decision analysis when using this sensor. Figure 3 shows an initial plan view of Jason's world. It consists of two connecting rooms, R60 and R70, containing various boxes. Jason, currently in R60, is designated by the symbol "J" followed by an arrow indicating his principal orientation. The symbol "W" designates a wall, while the "+" sign indicates a clearance border for navigation purposes. Jason's only current goal is "IN(J,R70,100)," i.e., Jason desires to be in the adjoining room with a utility value of 100 ergs. Thus, the accumulated cost of all his effort (both planning and execution) in accomplishing this goal should not exceed 100 ergs; otherwise Jason will have wasted his energy.

Now in this formulation, although "thinking" is assumed to be free of charge, every operation Jason can carry out in the real world is energy consuming. Table 2 shows the approximate cost of an application of each of the ILOs relevant to this

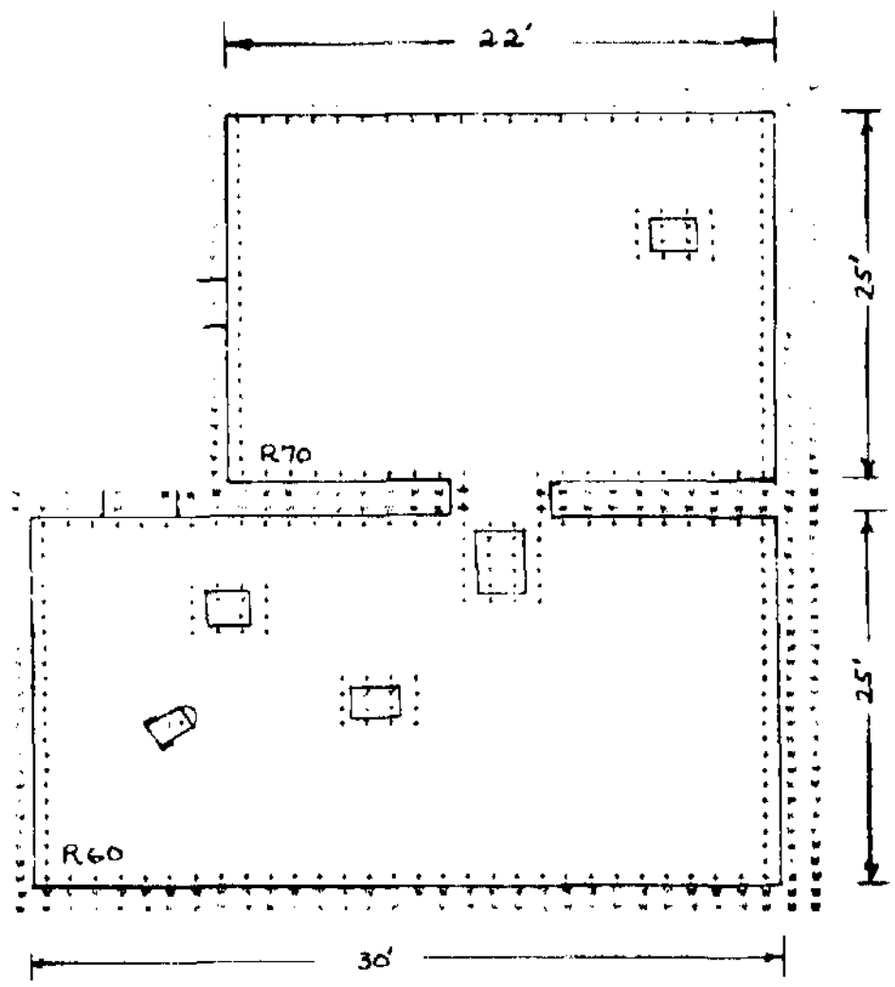


FIGURE 3 PLAN VIEW OF JASON'S WORLD

problem. Note that in general they entail a substantial start-up cost possibly plus an amount proportional to distance moved or turned. The proportionality constant, c, is on the order of 1/10 when distance, d(x,y), is measured in feet. Also note that "pushing" is twice as expensive as "moving." The preconditions and postconditions indicated are typical of a STRlPS-like[13] symbolic problem solver.

Under normal conditions the problem solver would yield the plan: (i) test for clear path to door D67 (connecting Rooms R60 and R70), (ii) Go to Door; (iii) Go through door. This could be accomplished for about 25 ergs, making it quite an attractive plan. However, in this formulation there is a box blocking the doorway, which would have caused "clearpath" to fail. Thus, the problem solver might instead generate the plan (i) Go tobox; (ii) Pushbox (outoftheway); (iii) Go to door; (iv) Go through door. This plan might executed for about 35-40 ergs, still making it " quite attractive.

However, in this particular world there is an additional complexity regarding the pushability of boxes. By definition, there are two general types of boxes: short and tall. Short boxes are not directly pushable because Jason's push bar would overhang them. Nevertheless, a short box could be pushed by means of a pushable tall box as shown in Figure 4. Furthermore, tall boxes come in two varieties: smooth and rough. Smooth boxes are

Actual costs are accumulated by calls to Low Level Operators (LLOs) which may vary depending on the circumstances.

Operator	Description	Approximate Cost (ergs)	Preconditions	Postconditions: Delete List	Postconditions: Primary Add List	Postconditions: Secondary Add List
1. Clearpath(a)	Ultrasonic Torch determines whether an obstacle blocks a straight-line path to point a.	1	Face(J,a)		Clearpath(a)	
2. Texture(B,p)	Ultrasonic Torch determines whether the texture of box B is rough or smooth with probability p.	1	Face(J,B) IN(J,R←$1) IN(B,R)		Texture(B)= Rough or Smooth	
3. Turn(t)	Jason turns t degrees.	$3+\frac{\alpha}{45}t$	Clear(t) Angle(J,θ←$1)	Angle(J,θ)	Angle(J,θ+t)	
4. Move(d)	Jason moves d feet.	$3+\alpha d$	AT(J,a←$1) Clearpath(a+d) IN(J,R←$1) IN(J+d,R)	AT(J,a)	AT(J,a+d)	
5. Goto(a)	Jason goes to point a in the same room.	$5+\alpha d(J,a)$	IN(J,R←$1) IN(a,R)	AT(J,$1)	AT(J,a)	
6. Gotodoor(D)	Jason goes to door D connecting the current room to another adjoining room.	$5+\alpha d(J,D)$	IN(J,R←$1) CONN(D,R,$1)	AT(J,$1)	AT(J,D)	
7. Gothrudoor(D)	Jason goes through door D.	5	AT(J,D) IN(J,R←$1) CONN(D,R,R'←$1)	IN(J,R)	IN(J,R')	
8. Push(B,a)	Jason pushes box B to point a.	$8+2\alpha d(B,a)$	Pushable(B) IN(J,R←$1) IN(B,R) IN(a,R) AT(B,b←$1) AT(J,b)	AT(B,b) AT(J,b)	AT(B,a)	AT(J,a)

Notes: d(x,y)= distance from x to y. $\alpha \simeq 1/10$. "$1" matches a single arbitrary constituent. "x←$1" means that x is bound to whatever value "$1" matches.

TABLE 2 INTERMEDIATE LEVEL OPERATOR (ILO) DESCRIPTIONS

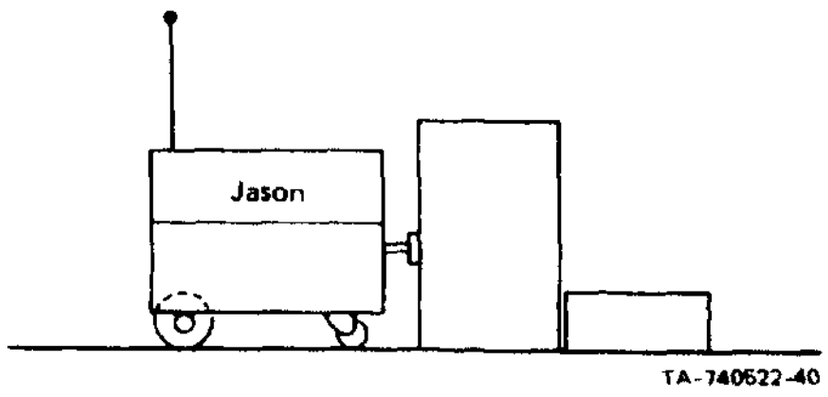


FIGURE 4 JASON USING A TALL BOX TO PUSH A SHORT BOX

pushable, while rough boxes are nonpushable (because they're too heavy). Now if the box blocking the doorway were tall, the preceding plan might be put into effect. However, it is known by Jason to be short. Thus, in such a case the existence of an available smooth tall box is a prerequisite to solving the problem. Now Room R60 contains not one, but two tall boxes. If Jason knew for a fact that at least one of the boxes were smooth and which one it was, the following plan might be generated: (i) Go to the smooth box; (ii) Push the smooth box to the low box (blocking the door); (iii) Push both boxes (out of the way); (iv) Go to door; (v) Go through door. Such a plan might be executed for about 60-70 ergs and corresponds closely to the original monkey and bananas problem. If both boxes were known to be smooth, Jason would choose to go to the box that minimized the combined cost of going and pushing. Note that the

assymetrlc cost of these operations would in general place a premium on the box closer to the low box. Since both boxes are approximately equidistant to both Jason and the low box, Jason is indifferent according to this criterion.

However, In this case Jason does not know a priori whether neither, one, or both tall boxes are smooth. The a priori probability distribution of rough and smooth tall boxes is assumed to be uniform. Jason has avallable through its ultrasonic range-ftnder, a texture operation he can perform to determine whether a box Is smooth with probability p, i.e.,

Prob(smooth|texture-smooth)=p
Prob(smooth|texture=rough) -1-p
Prob(rough |texture=smooth)-1-p
Prob(rough |texture=rough) =p.

Using common sense, if p is very close to 1, then a cost of 1 erg to measure texture is likely to be a worthwhile investment compared to the risk of going to a possibly rough box while the other one might have been smooth. On the other hand ae p approaches 0.5, the texture operation becomes increasingly unreliable and the marginal utility of the information obtained with respect to the cost of gathering it becomes smaller. Beyond some specified point of indifference as p approaches closer to 0.5, the cost of information gathering becomes prohibitive compared to Its value. Testing texture would actually be counterproductive, and the best strategy would be to choose a box at random, go to I t , and t r y to push I t . In the worst case for which a solution were possible, i.e., the first box tried was rough while the other was smooth, the cost might be between 80 and 110 ergs, therefore still worth trying.

Now Jason has some empirical statistics on the reliability of his range finder that indicate that p=0.95. What should he do? Texture operations are expensive and risky. Yet any action may be frought with peril. Jason turns to decision analysis to provide an answer. Figure 5 shows Jason's Decision Tree for the relevant portion of this problem. The results of the analysis are presented ln the next section.

Results of the Decision Analysis

The internal representation of the decision tree for evaluation purposes is as a two-page "SEETREE" program[14] which is very similar in appearance to Fortran IV. Each node, whether a decision node or a probability node, is defined as a separate function with its successor designated by the identifier "NEXT." The SEETREE program is translated into a standard FORTRAN IV program by a translator package operating over the ARPA Net at UCLA. Upon execution, the system generated a 56 node tree. Table 3 shows the "selected decision*' for various values of p. The system also automatically provides rollback values of a l l subsequent decisions in the tree and the minimum, maximum, mean, and standard deviation of rewards for the

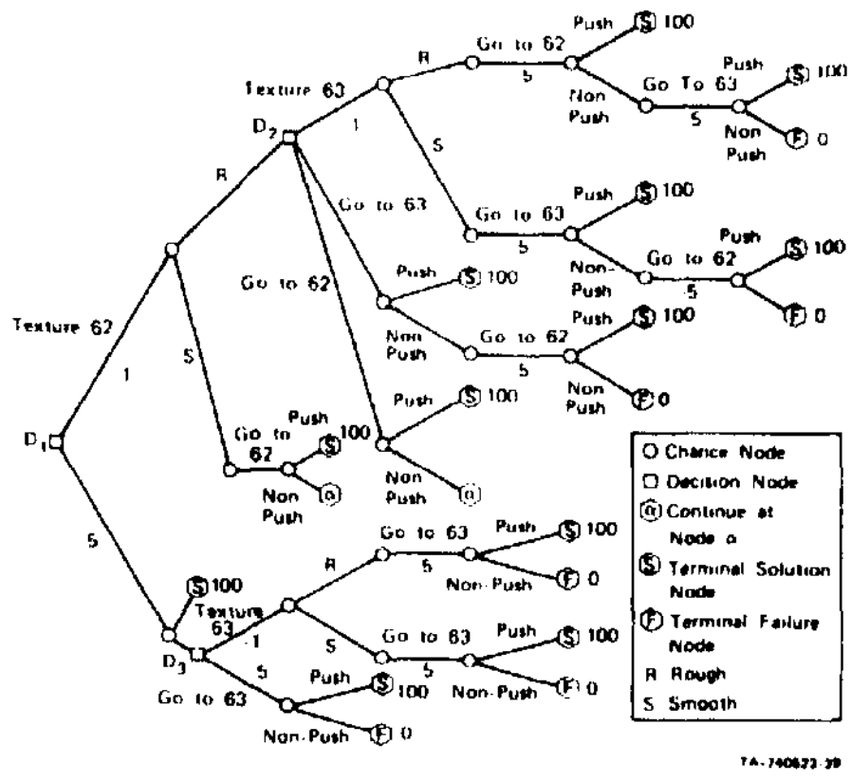


FIGURE 5 JASON'S DECISION TREE

lottery on the tree. A cumulative probability distribution of lottery values is also plotted automatically.

p	Selected Decision	Certain Equivalence	Certain Equivalence of Alternative Decision
0.80	GOTO Box62	67.500	67.250
0.90	GOTO Box62	67.500	67.500
0.95	TEXTURE Box62	67.625	67.500
1.00	TEXTURE Box62	67.750	67.500

TABLE 3 RESULTS OF DECISION TREE EXECUTION

Inspection of Table 3 reveals Jason's Indifference point to be p=0.9, since the certain equivalence for either decision is the same. Since Jason's historical data suggests a value of p-0.95, his optimal strategy should be to measure texture before going to either box. If the sensor reports back "smooth," then he should go immediately to that box. Otherwise he should go directly to the other box without further measurement, since further increase in the confidence of its true texture would not make a difference in Jason's subsequent behavior.

* Providing p<l. Of course, if p=l and the first box was discovered to be rough, then testing the texture of the second box would be useful: if it too were rough, Jason could definitively give up without moving. However, if there is even the slightest possibility that the sensor was in error (p<l), Jason would be compelled, lacking other alternatives, to verify a box's roughness empirically by going to it and trying to push i t .

```
/  ASSUMED INITIAL SITUATION  /

        /******  DESCRIPTION OF ROOM 60 AND ITS CONTENTS  ******/

THE LENGTH OF ROOM60 IS 30 FEET.
THE WIDTH OF ROOM60 IS 20 FEET.

DOOR1 IS AT (0, 12+1/2).  THE WIDTH OF DOOR1 IS 4 FEET.
DOOR2 IS OPEN.
  DOOR1A0 IS AT (-1/2, 12+1/2).  THE WIDTH OF DOOR1A0 IS 3 FEET.
  DOOR1A0 IS CLOSED.

BOX1 IS AT (5, 7).
BOX2 IS AT (-5, 5).
BOX3 IS AT (-5, 0).
THE LENGTH OF BOX1 IS 2 FEET.  THE WIDTH OF BOX1 IS 4 FEET.
THE LENGTH OF BOX2 IS 2 FEET.  THE WIDTH OF BOX2 IS 2 FEET.
THE LENGTH OF BOX3 IS 2 FEET.  THE WIDTH OF BOX3 IS 2 FEET.
THE WEIGHT OF BOX1 IS 0.25 TON.
THE WEIGHT OF BOX2 IS 1 TON.
THE WEIGHT OF BOX3 IS 1 TON.
THE ANGLE OF BOX1 IS 0 DEGREES.
THE ANGLE OF BOX2 IS 0 DEGREES.
THE ANGLE OF BOX3 IS 0 DEGREES.

        /******  DESCRIPTION OF ROOM 70 AND ITS CONTENTS  ******/

THE LENGTH OF ROOM70 IS 20 FEET.
THE WIDTH OF ROOM70 IS 20 FEET.

DOOR3 IS AT (0, -12+1/2).
  DOOR3A0 IS AT (-1/2, 11).      THE WIDTH OF DOOR3A0 IS 3 FEET.

THE LENGTH OF BOX4 IS 2 FEET.
THE WIDTH OF BOX4 IS 2 FEET.
THE WEIGHT OF BOX4 IS 1 TON.
THE ANGLE OF BOX4 IS 0 DEGREES.
BOX4 IS IN ROOM70.  BOX4 IS AT (7, 6).

START(J).

YOU ARE IN ROOM60.  YOU ARE AT (-10, -2).  THE ANGLE OF JASON IS 180 DEGREES.
                /  T E S T   R U N   O F   J A S O N  /
                START PLAN1 ROOM60.
```

FIGURE 6 INITIAL AND BOUNDARY CONDITIONS

```
PLAN 1.  GET TO ROOM 70:
    /  CHECK IF PATH TO DOOR IS CLEAR  /
    IF #CLEARPATH (ARM, ANY), (4, 10) THEN BEGIN
        GOTO ROOM70.
        END
    ELSE  / CHECK IF THE BOX THAT IS BLOCKING THE DOOR IS PUSHABLE  /
    IF #PUSHABLE BOX1) TRUE THEN BEGIN
        #PUSHTO BOX1 (5, 9)  ROOM60.
        GOTO ROOM70.
        END
    ELSE  /  GET A BOX WITH WHICH TO PUSH THE BLOCKING BOX  /
    IF #EXISTS (BOX2 SUCH  THEN BEGIN
        /  A SMOOTH BOX IS MORE LIKELY TO BE PUSHABLE  /
        IF #PUSHABLE BOX2) TRUE  THEN BEGIN
            #PUSHTO BOX2 (2, 9) ROOM60.
            #PUSHTO BOX2 (8, 9) ROOM60.
            GOTO ROOM70.
            END
        ELSE  /  CHECK IF LAST REMAINING BOX IS PUSHABLE  /
        IF #PUSHABLE BOX3) FOR  THEN BEGIN
            #PUSHTO BOX3 (2, 9) ROOM60.
            #PUSHTO BOX3 (8, 9) ROOM60.
            GOTO ROOM70.
            END
        END
    ELSE  /  TRY THE OTHER BOX FIRST  /
    IF #PUSHABLE BOX4) TRUE THEN BEGIN
        #PUSHTO BOX4 (2, 9)  ROOM60.
        #PUSHTO BOX4 (8, 9)  ROOM60.
        GOTO ROOM70.
        END
    ELSE  /  TRY LAST POSSIBILITY  /
    IF #PUSHABLE BOX2) TRUE THEN BEGIN
        #PUSHTO BOX2 (2, 9)  ROOM60.
        #PUSHTO BOX2 (8, 9)  ROOM60.
        GOTO ROOM70.
        END
#SPACE.
  END PLAN.
PLAN 1, TO ENSURE A CLEAR PATH
#ENSURE BOX2 SMOOTH.
  #ENSURE BOX3 SMOOTH.

GOODBYE.
```

FIGURE 7 PLAN1

Now, in the case where Jason goes to either the first or the second box and discovers it to be rough (i.e., non-pushable), he must then go to the alternative box in the hope that it might turn out to be smooth, regardless of what he thought it likely to be. Only if both boxes are directly proven to be rough can Jason give up with assurance.

Based on the above analysis, direct output from the Jason simulator is provided in Figures 6-11. Figure 6 provides a statement of the initial conditions. Comments are indicated between slashes. The English sentences appear just as they would be typed into the Jason terminal for interactive execution. They are translated into an internal grid model shown in Figure 3 by a simple natural-language parser based on a precedence grammar.

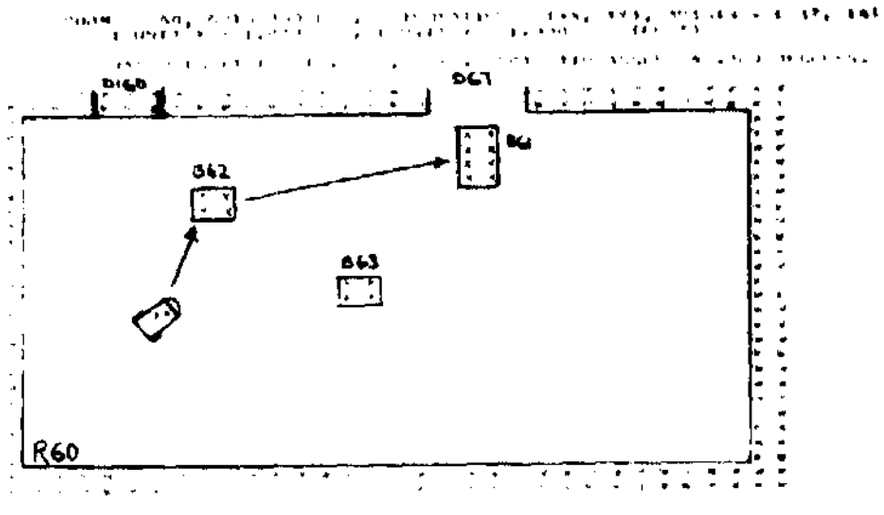


FIGURE 8 INITIAL STATE OF R60

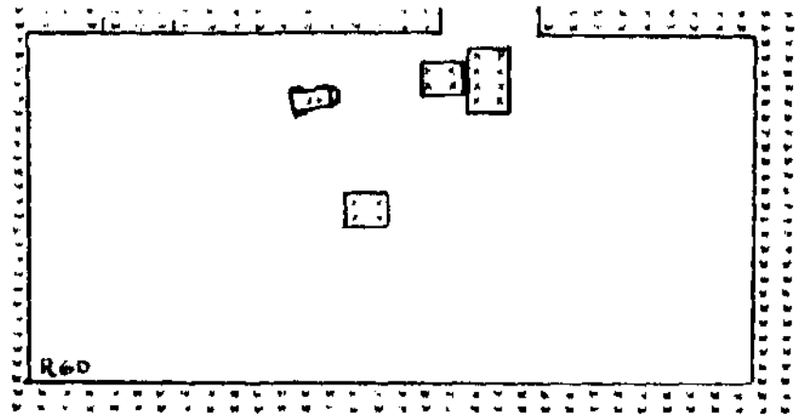


FIGURE 9 B62 TOUCHES B61

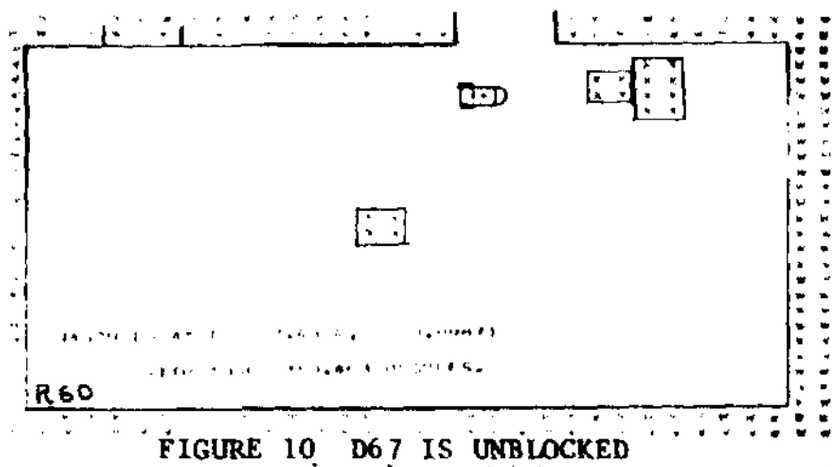


FIGURE 10 D67 IS UNBLOCKED

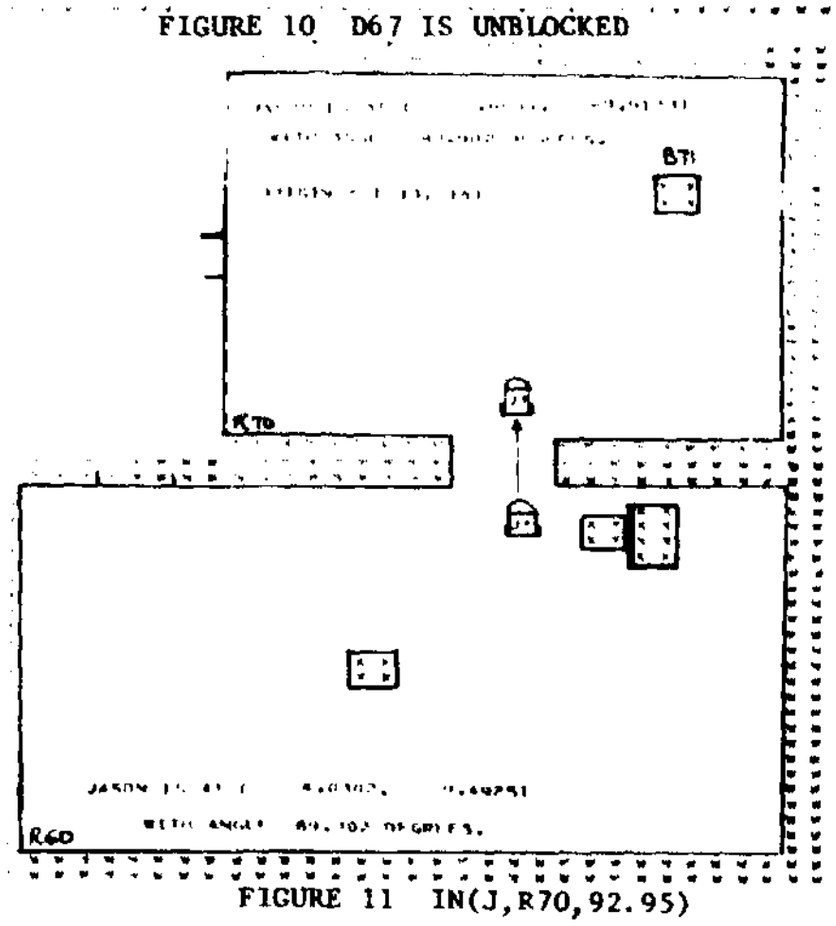


FIGURE 11 IN(J,R70,92.95)

Figure 7 shows the form of the plan output by the symbolic problem solver (comments inserted between slashes). PIanI is then executed interpretively. In this case under the assumption that the texture of Box62 is reported to be smooth and further that it is actually smooth, and therefore pushahle, the fina1 cost of the plan under simulated execution is 92.95 ergs. Since this value is within the specified utility of 100, Jason has achieved a net profit of over 7 ergs.

Figure 8 shows the initial state of R60 with the proposed trajectory marked by arrows ("+" signs have been suppressed.). In Figure 10 Jason has pushed both B61 and B62 out of the way. Finally, Figure 11 shows Jason having executed the Gotodoor/Gothrudoor portion ot his plan- In execution, this plan produced a string of 19 individua1 move and turn commands, each of which generates its own motor command or how Level Operator (LLO), resulting in a 50-page listing of grid positions. The simulated execution took approximately 3 seconds on the CDC-6400 and 7.6 seconds on the PDP-10. Based on prior experience, we expect that actual execution in tile real world will take on the order of 3 minutes.

Summary and Future Work

The major contribution of this paper is a demonstration, by means of a we]1 known example, of how decision analysis can be used to improve the decision-making capability of a robot under conditions ot uncertainty in its perceptual inputs. Now that Jason has been demonstrated over the ARPA Net, our next priority will be to link the decision analysis software with the Jason control software as an integrated system. We can then experiment dynamically with different boundary conditions as we11 as different levels of reliability tor texture measurements.

In the longer term we have outlined a number of objectives. Our intermediate goals include the development of software to handle (i) multiple (possibly conflicting) time-dependent goals, (ii) dynamic tracking (of a cooperative agent), and (iii) dynamic real-t ime collision avoidance during navigation, including the evasion of active agents such as slowly moving people in a crowded corridor. Our long term goals include the application of a Jason-like robot to factory or warehouse work where the environment is fairly well controlled.

The Berkeley robot project is still in the midst of design and construction. Many devices have been designed, tested, and installed. A considerable amount of software has been written and debugged. Other devices and programs are still in the development stage. It is planned that a fully-integrated robot vehicle will again be operational in the near future. Once operational, Jason will be used as a test bed for the development of future, general-purpose mobile robots with even greater sophistication and intelligence.

Acknowledgments

The authors would like to Ihank the Department of Electrical Engineering and Computer Sei - ences of the University ot California at Berkeley for its continuing support of the Jason Project. In particular, Prof. David Hodges and Mr. Leonard Baldwin of the E.E.C.S. machine shop deserve special mention. We also wish to acknowledge the assistance of Dr. Ramon Zamora oi the Decision Analysis Group at Stanford Research Institute.

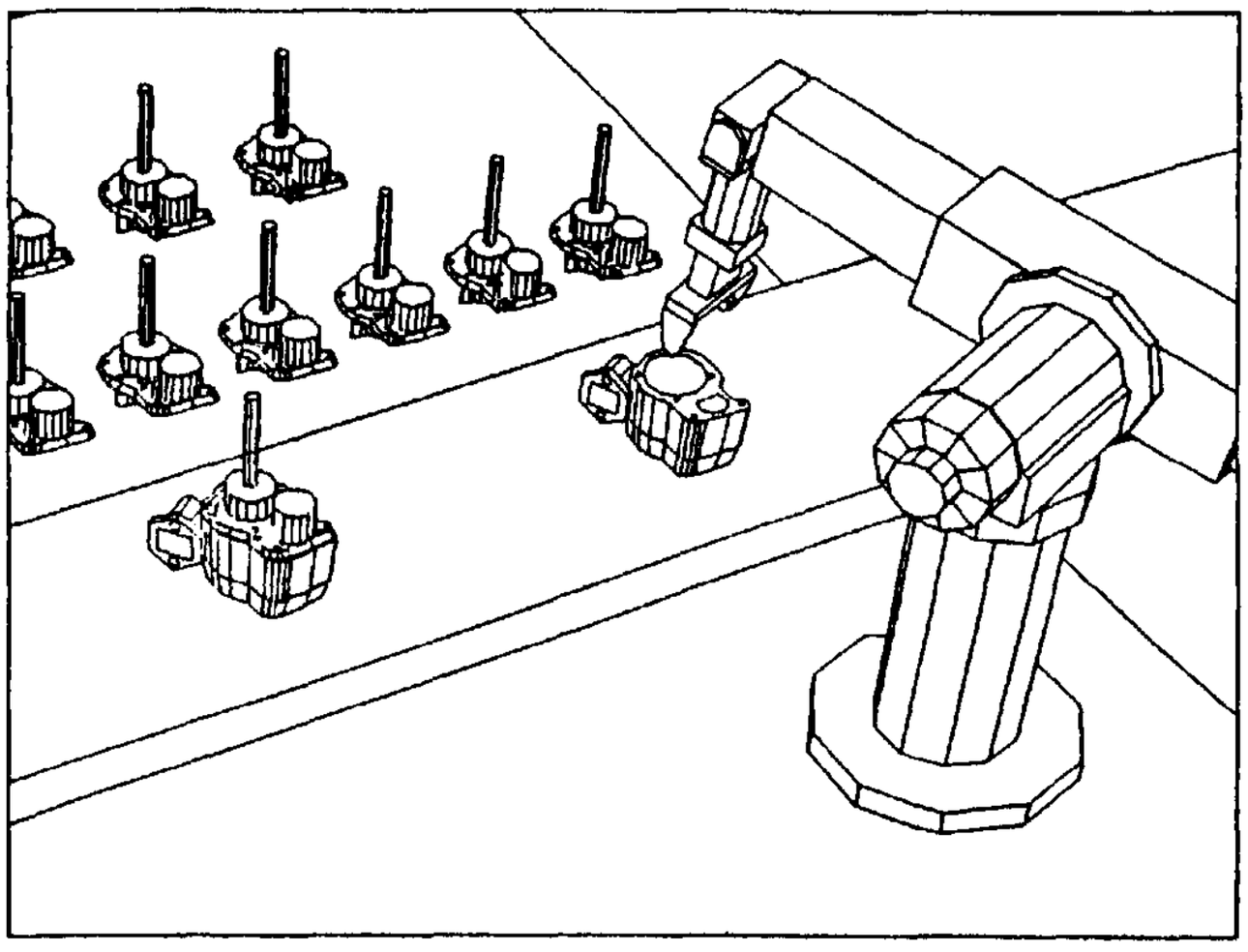
Bibliography

1. Michael H. Smith and 1.. Stephen Coles, "Design of a Low-Cost General-Purpose Robot," Proc. of the IJCAI-73. pp. 324-335 (August 1973).
2. Alan Robb, "A Design of a Communications Controller for Jason, the Berkeley Robot," EECS 292H Project Report, The University of California at Berkeley (June 1974).
3. Paul L. Sinclair and Ralph P. Sohek, "Jason Reference Manual," EECS 292H Project Report, The University of California at Berkeley (June 1974).
4. George W. Ernst and Allen Newell, GPS: A Case Study in Generality and Problem Solving (Academic Press, 1967).
5. John H. Munson, "Robot Planning, Execution, and Monitoring in an Uncertain Environment," Proc. of the IJCAI-71, pp. 338-349 (September 1971).
6. John McCarthy, "Situations, Actions, and Causal Laws," Memo No. 2, Stanford University Artificial Intelligence Project, Stanford, California (July 1963).
7. L. Stephen Coles, "An Experiment in Robot Tool Using," Proc. of the IEEE Systems. Man, and Cybernet ics Society International Conference (Pittsburgh, Pennsylvania, 1970).
8. Drew V. McDermott, "Assimilation of New Information by a Natural Language-Understanding System," M.S. Thesis, TR-291, MIT Artificial Intelligence Laboratory (February 1974).
9. Jerome A. Feldman and Robert A. Sproull, "Decision Theory and Artificial Intelligence 11: The Hungry Monkey," Technical Report No. 2, Computer Science Department, University of Rochester, New York (November 1974).
10. Warner North, "A Tutorial Introduction to Decision Theory," IEEE Transactions on Systems Sciences and Cybernetics, Vol. SSC-4, No. 3, pp. pp. 200-210 (September 1968).
11. Ronald A. Howard, "The Foundations of Decision Analysis," IEEE Transactions on Systems Sciences and Cybernetics. Vol. SSC-4, No. 3, pp. 211-219 (September 1968).
12. Herman Chernoff and Lincoln E. Moses, Elementary Decision Theory (John Wiley and Sons, New York, 1959).
13. Richard E. Fikes and Nils J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving in Problem Solving," Artificial Intel ligence, Vol. 2, Nos. 3/4, pp. 189-208 (Winter 1971).
14. Ramon M. Zamora and Ellen B. Leaf, "Tutorial on the Use of the SRI TREE Language System" Technical Memorandum, Stanford Research Institute, Menlo Park, California (December 1974),

AN OVERVIEW OF AL, A PROGRAMMING SYSTEM FOR AUTOMATION

Raphael Finkel, Russell Taylor, Robert Bolles,
Richard Paul*, Jerome Feldman*
Stanford Artificial Intelligence Laboratory
Stanford, California USA

June 3. 1975



* Jerome Feldman is now at the University of Rochester. Richard Paul is now at the Stanford Research Institute.

This research was supported in part by the National Science Foundation under contract No. GI-42906 and in part by the Advanced Research Projects Agency of the Office of Defense under Contract No. DAHC-15-73-C-0435. The views and conclusions in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the funding agencies.

ABSTRACT

AL is an high-level programming system for specification of manipulatory tasks such as assembly of an object from parts. AL includes an ALGOL-like source language, a translator for converting programs into runnable code, and a runtime system for controlling manipulators and other devices. The system includes advanced features for describing the motions of manipulators, for using sensory information, and for describing assembly algorithms in terms of common domain-specific primitives. This paper describes the design of AL, which is currently being implemented as a successor to the Stanford WAVE system.

AN OVERVIEW OF AL

This short paper cannot cover the subject of AL in depth; a complete discussion may be found in *AL, A Programming System Jor Automation,* Stanford Artificial Intelligence Laboratory Memo AIM-243, Stanford University Computer Science Department Report STAN-CS-74-456, by the authors of this paper.

INTRODUCTION

The development of robot manipulators such as the "Unimate" has led to the belief that these tools are in some way general-purpose devices and that they might be programmed like a computer As a general-purpose programmable device, the robot manipulator provides a possible answer to the need for automation of assembly in batch manufacturing industries where small production runs rule out the use of special-purpose equipment.

We are implementing a system called AL for small scale batch manufacturing where setup time is the key factor. We rely on a symbolic database and previously-defined assembly primitives to minimize programming time. The system is capable of high-level planning and intelligent interpretation of user-defined primitives. The principal aim of this work is not to provide a factory floor programming system but rather to design a language which will be a tool for investigating the difficulty, necessary programming time, and feasibility of writing programs to control assembly operations.

PHILOSOPHY OF DESIGN

DATA AND CONTROL STRUCTURES

The principal mode of input to AL is textual, as opposed to spoken or manual (joystick). There are levels of complexity which are much more readily transmitted from man to machine through an interlace of symbolic text, for example, simultaneous motions of two arms and termination and error conditions are more likely to be unambiguously described through the medium of text because a textual language can provide a consistent framework for such intuitive ideas. Non-textual forms of input for defining target locations and suggesting arm trajectories to avoid collisions are most useful when applied in conjunction with a program text which supplies the overall intent of the programmer. The supervisor level of AL is simple enough to allow natural teaching by showing; it should be easy to interface such devices as joysticks and vocal input into AL, although we do not intend to do so at present.

Experience with languages like SAIL and WAVE has shown that text macros are a useful feature; they reduce the amount of repetitive typing. AL has a general-purpose text macro system interfaced into the scanner and parser.

The datatypes available include those types necessary to refer to one-dimensional measures (like distance, time, mass) and three-dimensional measures (like directed distance, locations, orientations). Arithmetic operators are available not only for the standard scalar operations like multiplication and addition, but also for such operations as rotation and translation.

Provision is made for simultaneous execution of several processes This allows calculation and arm motion to take place simultaneously; several manipulators can be in independent or coordinated motion.

MOTION SPECIFICATIONS

Experience with WAVE has shown that calculating trajectories for manipulators is desirable but time-consuming. Trajectory calculations, together with other calculations which need only be performed once, are done at compile time. This allocation of effort drastically reduces the computing load at execution time and eliminates wasteful recomputation every time a sequence of actions is executed.

A wide range of exceptional conditions can occur during the motion of a manipulator. Appropriate action must be taken as soon as any of these occurs, for example to start up a new concurrent process or to notify the user. Therefore, AL allows the flexible specification of conditions to be monitored during motions (and during execution of blocks of code in general) and what to do in the case that a tested condition occurs

USE OF A PLANNING MODEL

Since locations are not known exactly during the planning of a trajectory, there is a clear distinction between planned values and runtime values. Planned values are used for trajectory calculation; at runtime, trajectories are modified if necessary to account for

any discrepancies. The planned values are therefore a database on which trajectory calculations are computed.

Assembly tasks require that one object be affixed to another. We model this by having a semantic attachment between objects, if two objects are affixed, and one moves, the second one should move accordingly, that is, its planning value should be properly modified. The planning model includes information on attachments of objects. The affixment concept carries over to the runtime system, which does the equivalent modifications of the actual values. This saves the user the tedious bookkeeping operations required to determine where an object is after its base has been moved.

More generally, the compiler maintains a wide variety of information about expected runtime states. This includes information like the accuracy within which the planning value is known, how heavy an object is, how many faces it has on which it can rest, how wide the fingers of an arm should open to grasp it. This information may come from several sources, including explicit assertions by the user and built-in knowledge about the system hardware. AL has a general framework for representing and using such knowledge.

USE OP DOMAIN-SPECIFIC KNOWLEDGE

The system will eventually have enough domain-specific knowledge to allow programs to be written in terms of common assembly operations, rather than exclusively in terms of detailed single motions. At the simplest level, this involves a library of common assembly macro-operations that can be conditionally expanded to perform particular subtasks. Beyond this, we foresee an interactive system that can take a "high level" description of an assembly algorithm and fill in many of the detailed decisions required to produce a consistent and efficient output program.

A user will be able to specify different parts of a task at various levels of detail. The system is designed to accept explicit advice telling exactly how some particular subtask is to be accomplished. This is especially important for early versions of AL, which are not likely to be very "smart" and will therefore require a fair amount of explicit help. Other parts can be described by assertions which specify prerequisites and effects. The system will then complete the program incorporating the advice and satisfying the other assertions. The system can show the user how it is filling in the details to produce an output program, and why. This is very important both for debugging and for explaining to the user any requests for advice that it must make

THE RUNTIME SYSTEM

The calculation of trajectories is time-consuming but not time-critical; servoing of devices is time-critical but not especially time-consuming. Therefore, the compiler is written in a high-level language, SAIL, which runs on a large timeshared computer (a POP-10) and the runtime system is designed to run on a dedicated minicomputer (a PDP-11/45).

The runtime system supports simultaneous execution of many processes. Several manipulators or devices might be running simultaneously, and each motor requires a separate process; several condition monitors might be active; several code segments (doing, perhaps, calculations) might be simultaneously active. Those processes which are dealing with real-time devices (joint servos and condition checkers) must be guaranteed service at regular intervals; the computation processes can fill in any time gaps.

The wide range of conceivable tasks implies that pure hardware servoing does not in general suffice. The reason for this is that hardware servoing restricts use to one of a small number of servo modes (typically position, velocity, or force), and has no provision for motions of accommodation or motions whose modes might change in midstream due to some software-detectable condition. Pure hardware servoing could not be readily modified to account for new feedback devices or methods. A philosophy of *software servoing* has these advantages; It is possible to program the manner in which feedback is to be used, to interface new types of sensors, to modify the servo while the arm is in motion, to supply the driving program with information concerning the success of the motion as well as to keep it up-to-date on the arm status. It also allows coordination of several arms, with one acting as a master and the others following. Hardware servoing would not save computation since the computer would need to perform an equivalent servo calculation in order to understand what the manipulator is doing.

GENERAL SYSTEM OUTLINE

HARDWARE

Currently two Stanford Electric Arms, built by Victor Scheinman [Scheinman], are available. They are called YELLOW and BLUE. Each has six joints and a hand that can open and close. The joints are controlled by electric motors; each joint has both position and velocity feedback. Motor drives are sent from the computer to the arm via a digital-to-analog converter (D-to-A); feedback signals are routed through an analog-to-digit at converter (A-to-D) back to the computer.

Various other devices are designed and implemented as needed. We use tools, jigs and special markings for several purposes: to render a task possible (an example is the arm itself), to improve efficiency (a mechanical screwdriver), and to overcome some of our sensory and mechanical limitations (a screw dispenser)

SOFTWARE

See figure I for a picture of the system,

The SUPERVISOR is the top *level* of AL ft runs on the timesharing computer and provides an interface between the user and the other parts of the system: 1) listening to the user's console and interpreting simple command language input; 2) controlling the compiler, starting it and relaying its error messages back to the user; 3) signalling the loader when it is necessary to place compiled code into the mini; 4) handling the runtime interface to the mini.

The USER sits at a console and makes requests of AL. These fall into several categories; compilation, loading, execution of programs, debugging of code, requesting of status information, asking for immediate arm motion, saving and restoring the state of the world at safe points, requesting explanation of certain compiler *decisions. There* are two different consoles at which a user can sit: one is connected to the timesharing computer, through which he *can* speak to the supervisor and all the parts of AL residing on the timesharing computer; the other is *connected* to the mini, and through it the user can investigate the runtime system and cause modifications.

The COMPILER reads AL programs from files (or, optionally, directly from the user's console) and produces load modules, The compiler is divided into three phases: The PARSER, which produces parse trees of the program, the EXPANDER, which expands those parse trees by replacing high-level primitives with low-level primitives, and the TRAJECTORY CALCULATOR and CODE GENERATOR, which creates the output files

The LOADER takes the load modules prepared by the compiler and enters them into the mini's runtime system. Address relocation and linking are done at this time. The loader also sets up the data area in the runtime interface in the timesharing computer; these data include output strings, procedure linkages, and information necessary for diagnostic purposes during runtime. Loading is often done in a partially incremental fashion, installing new code following previously loaded code.

The RUNTIME INTERFACE, which resides in the timesharing computer, is charged with initiating the mini program, fielding procedure calls from the running program to procedures on the timesharing machine, returning values from these procedures, and fetching values from the mini for debugging purposes. The interface has the power to interrupt the execution of the program and to modify the status of *the* runtime system, for example, by patching in additional programs or modifying the values of some variables. This allows the user to control the program through the timesharing computer.

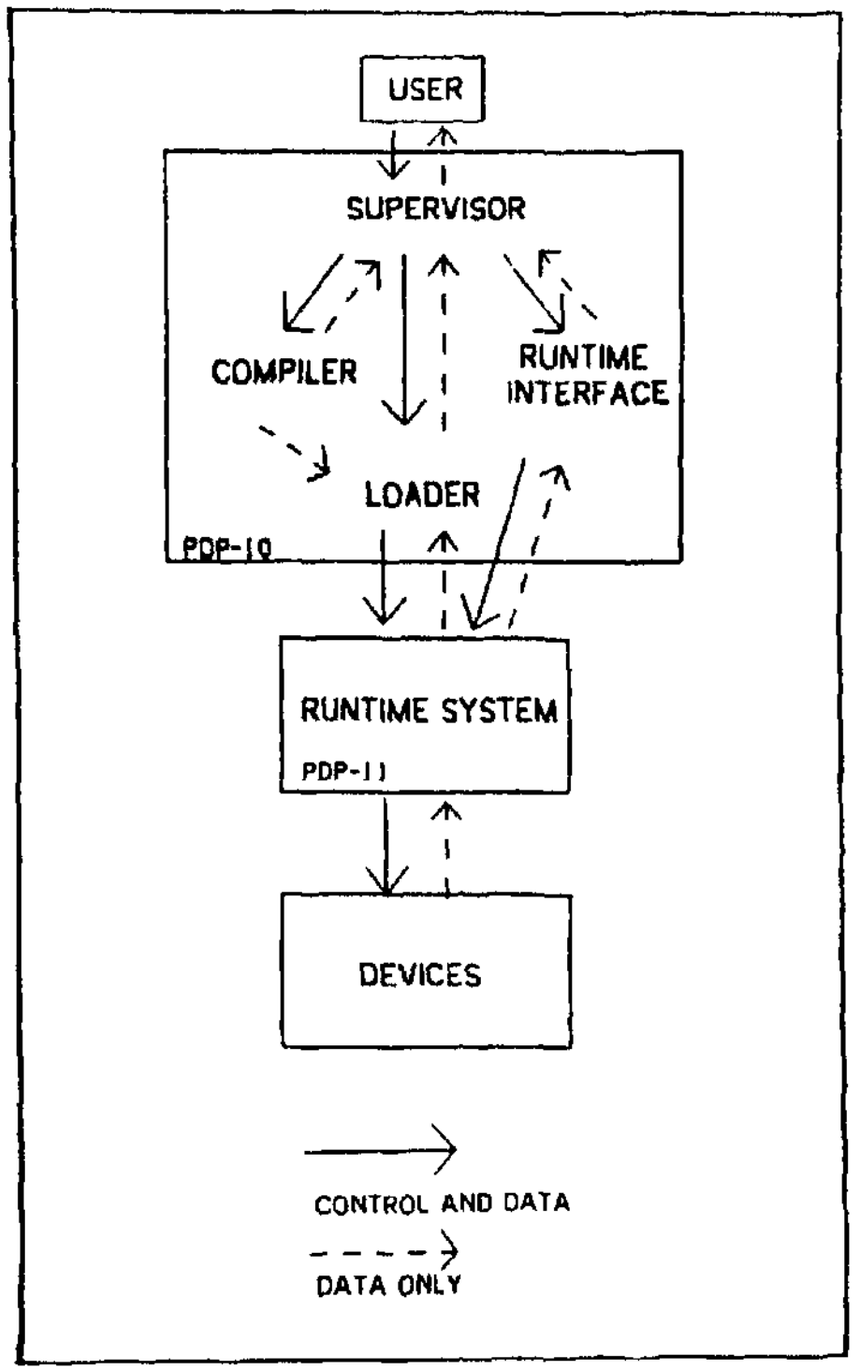


FIGURE 1.

The RUNTIME SYSTEM is the set of programs which reside in the mini. This system includes kernel programs for time-slice cpu sharing and process control and a set of dynamically created processes. These are of three basic types: a) An INTERPRETER examines the code prepared by the compiler and executes the numeric computations requested. When a move is to be started, the interpreter creates a servo for each joint and waits until all these servos are finished, b) A SERVO handles the motion of one moving joint, c) A CONDITION-MONITOR repeatedly examines certain conditions (whatever the programmer has specified). If it should discover that its condition has occurred, it creates an interpreter to take appropriate action. The runtime system also includes routines for communication with the runtime interface in the timesharing computer.

AN EXAMPLE

A simple task dealing with the objects shown in Figure 2 illustrates some of the features of AL;

1. Pick up the bracket with the YELLOW arm and position it next to the beam so that the holes line up,
2 Pick up the bolt with the BLUE arm,
3. Fasten the bracket to the beam by inserting the bolt in the holes,
4. Return the arms to their park positions.

Some of the features to be demonstrated are: the affix structure, reference frames, dimensions, and multi-processing. We demonstrate these capabilities by presenting a highly commented AL program to accomplish the task stated above. Except for the missing macro bodies, this program is complete; it could perform as indicated.

AL is a multi-level programming language; at one extreme the user can write detailed, system-like programs and at the other he can describe the tasks and any partial ordering among them and let the system determine 'necessary details. Our example is written in an intermediate level. In particular, it assumes that there are several general-purpose macros and routines which understand how to GRASP and RELEASE things and carry out a NORMAL_SEARCH to insert something \nVo a hole

Capitalized words in the example are key words wilhin AL. Lower case words are user-defined identifiers. Comments are surrounded by curly brackets.

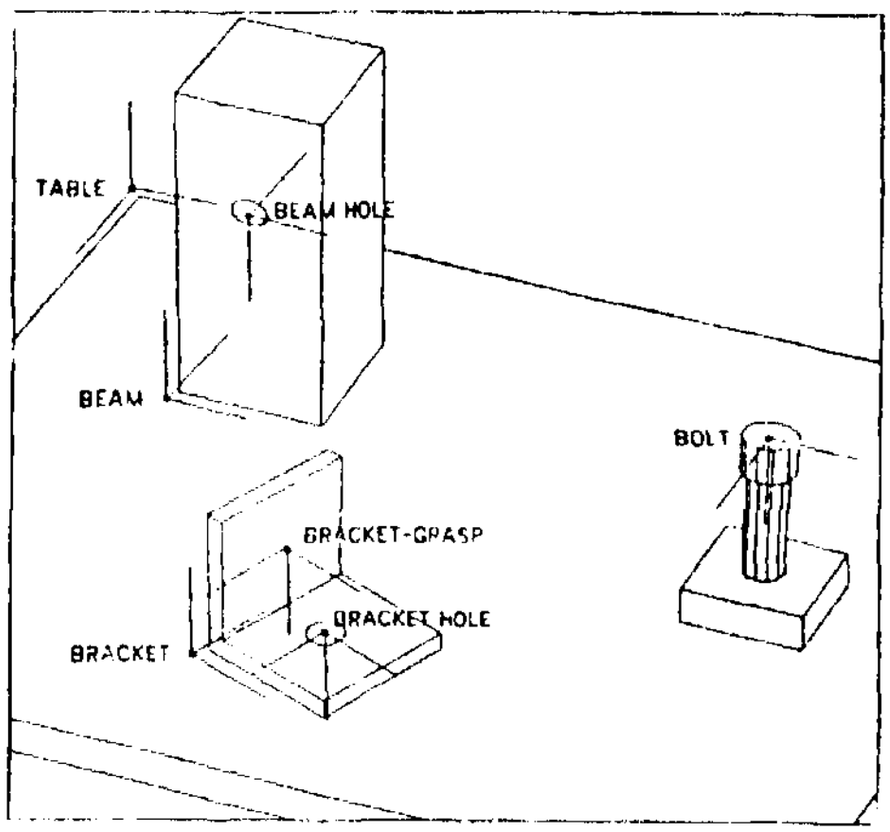


FIGURE 2.

```
whole_task  BEGIN
```

{First declare the necessary FRAMEs and describe how they are initially related A FRAME is a coordinate system It has two components, the location of the origin (a distance VECTOR) and the orientation of the axes (a ROT) Frames are typically used to describe objects and important features of objects. There are several predeclared frames in AL. STATION is the frame which represents the work station's frame of reference Each hand available to the system also has a frame variable, whose value (continually updated) is the position of that hand. Currently there are two such frames YELLOW and BLUE

The attach structure representing the initial world is shown in Figure 3 The arrows indicate how the movement of a frame affects other frames If a frame at the tail of an arrow is moved (by the arm, visually updated, etc) the frame at the head of the arrow will be automatically updated The double arrows are the results of RIGIDLY AFFlXmg one frame to another)

```
FRAME beam, beam_hole,        FRAME bolt,
FRAME bracket, bi acketjiole, bracket_grasp;
beam - FRAME(ROT(Z, 90*DEC), VECTOR(I0.6,0)>;
```

{The beam is expected to be positioned at (WjSP) in *the station'^ coordinate system (the default unit for distance measurements is centimeters) and rotated 90 degrees about the station's Z vector AL knows about dimensions like DEC for degrees. Dimensions are adjuncts to variable types, new ones can be defined in terms of old ones }*

```
beam_hole *- beam*TRANS(ROT(X, 90*DEG).
                VECTORO.O,?)),
```

{The TRANS represents the position of the beam_hole with respect to the beam. The premultiplication by the frame beam positions the beam^hde in the station's coordinate system)

```
AFFIX beam^hole TO beam,
```

{As shown in figure 3)

```
ASSERT FORM(DEPROACH, beamjiole.
        TRANS(NILROT,VECTOR(0,0,-3));
```

{Trajectories consist of a path from the current position, through a departure point, possibly through some via points, through an approach point, and finally to the destination The primary use of via points is to avoid collisions during the motion All FRAMEs have a DEPROACH TRANS associated with them TRANSes are essentially the same as FRAMEs Whenever leaving (or moving to) a FRAME the standard departure (or approach) used is that

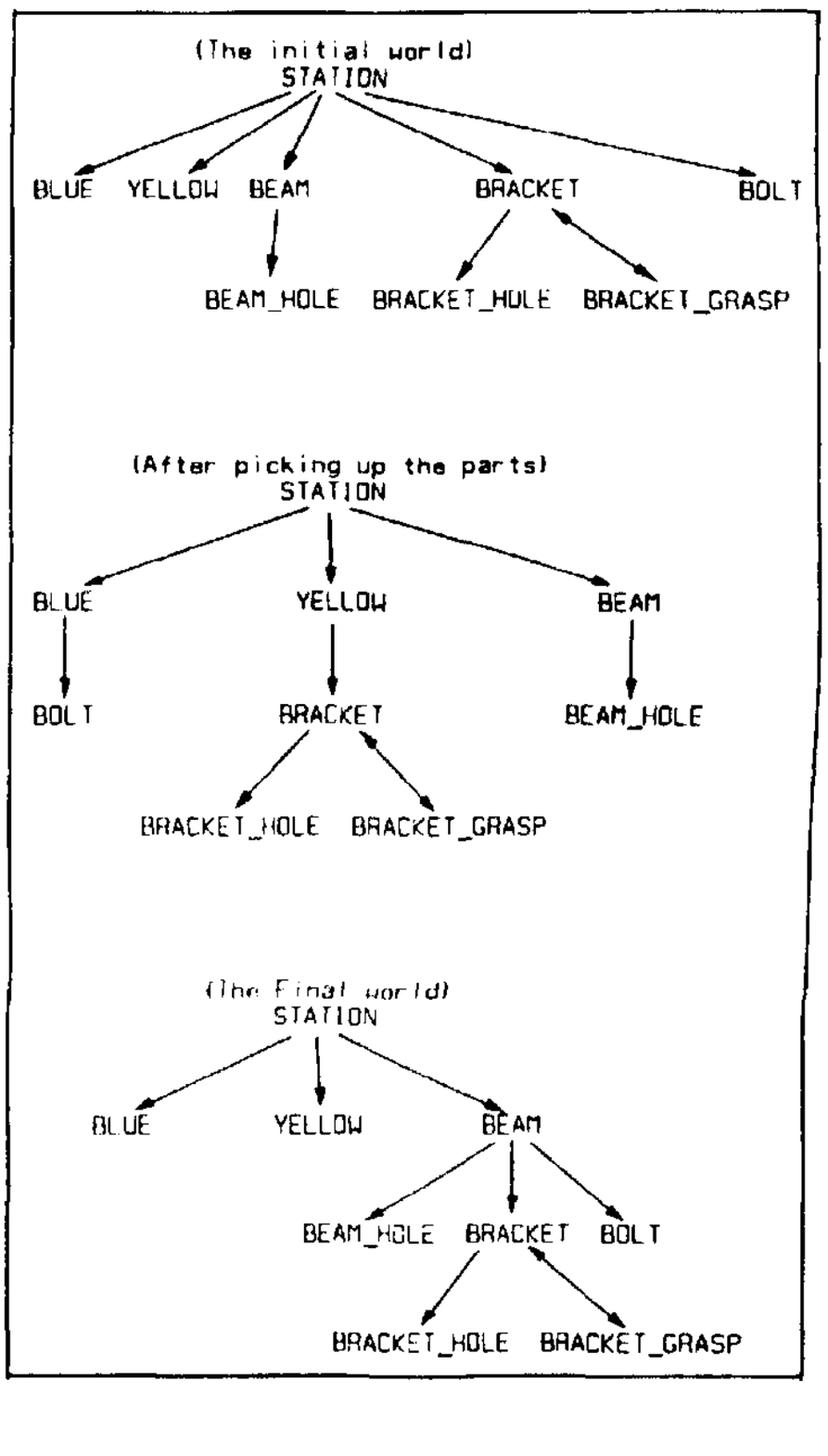


FIGURE 3.

FRAMEs DEPROACH TRANS The station has a DEPROACH TRANS which is three inches above it Whenever one FRAME is AFFlXed to another, by default the former takes on the latter's DEPROACH The result of this ASSERT is that the arms will approach the beam hole from the side instead of from above.}

```
bracket - FRAME(ROT(Z.45*DEG).
              VECTOR(20,140));
bracket„hole - bracket*
      TRANS<ROT(X.180*DEG).VECTOR(3,3,0));
AFFIX bracket.hole TO bracket;
bracket_grasp <-- bracket*
      TRANS(ROT(X,180*DEG)IVECTOR(0,3,3));
AFFIX bracket_grasp TO bracket RIGIDLY;
```

{The RIGID A FFIX ment insures that a change of bracket_grasp will automatically change bracket, which in turn will automatically change bracket-hole. This is quite convenient if the position of the whole 'object' is being updated by one grasping position (ie. bracket_grasp))

```
bolt - FRAME(ROT(Z.90*DEG)*
       ROT(X,I80*DEG),VECTOR(16.30,0));
```

{The rotation portion of the FRAME has been specified as a composition of two primitive rotations.)

```
DEFINE OZ - "72.007789*DYNES"
DEFINE INCHES - "254*CM",
```

{Some of the standard macros are defined next}

```
DEFINE grasp
     (TRANS speciaLdeparture.speciaLapproach,
      FRAME ATOM the.arm (DEFAULT YELLOW).
      FRAME object,grasp_point,
            thing_object_af fixed _to;
      DISTANCE SCALAR opening_before_departure,
            openmgJor_approach(DEFAULT15*CM),
            thickne$$(DEFAULT O.UINCHES))
      - " {body of macro goes here) ",
```

{The expansion of such a macro can depend upon the supplied arguments, the DEFAULT arguments, and any values in the current planning model.)

```
DEFINE release
     (FRAME ATOM the_arm(DEFAULT YELLOW);
      FRAME the_object,the_new_parent,
      DISTANCESCALAR the.openmg
                  (DEFAULT I5*CM))
      - M {body of macro goes here) ";
DEFINE normal_search
     (FRAME ATOM the_arm(DEFAULT YELLOW);
      DISTANCE SCALAR mcremem(DEFAULT .3*CM),
      distance_iwd,
      FORCE SCALAR stoppingjorce,
      SCALAR number_of_tnes(DEFAULT 9))
      • " {body of macro goes here) ";
```

{This would include some automatic error recovery and a call to the operator if something drastic goes wrong.}

```
COBEGIN
```

{This COBEGIN-COEND construction describes two independent subtasks (one for YELLOW and one for BLUE) which can be executed in any order determined by the runtime system, in parallel or serially. This, of course, assumes that the two arms work in completely separate parts of the workstation so there is no possibility of a collision}

```
ypickup BEGIN {pick up the bracket with yellow}
graspOobject- bracket ,grasp_point-bracket_gr asp,
    opening_for_approach-3*CM);
```

{Only the necessary parameters need to be specified. By default the YELLOW arm will be used and there will be no special approaches or departures. One effect of grasp is to AFFIX the object to the arm)

```
MOVE bracket.hole TO beam.hole +
   VECTOR(0,0,- 3) WRT beam.hole,
```

{The YELLOW arm (since it holds the bracket to which bracket_hole is AFFlXed) positions itself so that the bracket_hole lines up with the beam_hote, but is 3 cm away from the beam_hole The WRT operator is one way of describing a vector within a frame of reference other than the stations.}

```
MOVE YELLOW TO • •
      VECTOR(0,0,6) WRT beam.hole
   ON FORCE(Z WRT beam_hole)>50*OZ DO
      STOP YELLOW
   ON ARRIVAL DO
      ABORT("*ERROR* bracket went too far");
```

{The • represents the current position of the arm The arm moves 6 cm in Z relative to the beam_hole's frame The purpose of this move is to push the bracket up against the beam If the beam is there, the arm toill sense an opposing force. If not, the arm will succeed in moving forward the prescribed 6 cm. In order to check for these possibilities, two condition monitors have been included with the MOVE statement. The first one monitors the force and stops the arm if the force exceeds 50 ounces (which means everything is ok). The second one is an interrupt type condition If the arm successfully carries out the complete MOVE, this monitor is awakened, the message is printed, and control is given to an pperator.)

```
   END ypickup;
   bpickup. BEGIN {pick up the bolt with blue]
   grasp(the_arm-BLUE,the_object-boU,graip_point-bolt,
      opening_forapproach=3*CM);
   END bpickup,
COEND;
```

{Figure 3 shows the world after the parts have been picked up.)

```
MOVE bolt TO beamjiole+
      V£CTOR(0,0,-5.3) WRT beam_hole,
```

{The BLUE arm positions itself so that the bolt is lined up with the beam_hole and its tip is 3 cm away from Che outside of the bracket_hole J

```
normaUearcMBLUE, 2*CM, l.6*CM, 60#OZ. 9);
```

{This pushes the bolt through the bracket _hole and partly into the beam_hole If control continues past this statement the bolt is assumed to be partly in the hole)

```
MOVE BLUE TO * *
      FRAME(ROT(Z,90*DEC),VECTOR(0,0,4))
   ON FORCE(Z WRT BLUE) > 60*OZ DO
      STOP BLUE,
```

{This pushes and twists the bolt into the hole When the force exceeds 60 ounces, the bolt is assumed to be completely seated in the hole There is no check to make sure the bolt seats properly)

```
COBEGIN
   parky BEGIN prelease the bracket and park)
   reIea$e(the_object-bracket,
      the_opening-3!i:CM,the_new_parent-beam);
   MOVE YELLOW TO YPARK,
   END parky.
   parkb BEGIN {release the bolt and park}
   releasedhe_arm-BLUE.the_object-bolt,
      the_opening-3*CM.the_new_parent-beam);
   MOVE BLUE TO BPARK.
   END parkb.
COEND.
END wholeaask
```

CONCLUSIONS

AL is important for several reasons. It shows what sort of considerations are necessary for flexible control of mechanical manipulation. It demonstrates the feasibility of programmable assembly. It provides a research tool for investigation of new modes of software servoing, assembly primitives, arm-control primitives, and interactive real-time real-world systems.

AL is currently limited by the lack of certain features which would make it more competent. Many of these have to do with the fact that feedback is used only in a threshold way; either a monitor triggers or it does not. Fine control of the arm would be enhanced by more sensitive force-sensing elements on the hand and a means of programming accommodating, non-threshold response to this sensory input. Visual feedback should be implemented to provide better positioning capability, error detection, and error recovery Moving assembly lines imply that AL should be able to understand motions which it does not cause directly through manipulation; objects should have a dynamic capablility. Collision detection and avoidance remain difficult issues. AL would be more error-free if the trajectory calculator could ensure that the arms *ne^er* interfere with each other or with objects in the current world.

BIBLIOGRAPHY

[Bolles and Paul] R. C. Bolles, R. Paul, *The Use of Sensory Feedback in a Programmable Assembly System*, Stanford Artificial Intelligence Project, Memo No. 220, October 1973.

[Finkel] R. Finkel, R. Taylor, R. Bolles, R. Paul, J. Feldman, *AL, A Programming System for Automation*, Stanford Artificial Intelligence Project, Memo No. 243, November 1974.

[Inoue] H. Inoue, *Force Feedback in Precise Assembly Tasks*, Massachusetts Institute of Technology A. I. Memo No. 308, August 1974.

[Nevins] J. L. Nevins, D. E. Whitney, S. N. Simunovic, *System Architecture for Assembly Machines*, The Charles Stark Draper Laboratory, Inc., Memo No. R-764, November 1973.

[Paul] R. P. Paul, *Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm*, Stanford Artificial Intelligence Project, Memo No. 177, March 1973.

[Rosen] C Rosen, et. al., *Exploratory Research in Advanced Automation*, Stanford Research Institute Report, December 1973.

[Scheinman] V. D. Scheinman, *Design of a Computer Manipulator*, Stanford Artificial Intelligence Project, Memo No. 92, June 1969.

[Will] Peter M. Will and David D Grossman *An Experimental System for Computer Controlled Mechanical Assembly*, IBM Research Report RC 4922, Yorktown Heights, New York. July, 1974.