February 1972

# ROBOT RESEARCH AT STANFORD RESEARCH INSTITUTE

by

Bertram Raphael

Second of two lectures for JITA (Japan Industrial Technology Association International Symposium on Pattern Information Processing Systems, Tokyo, March 6-17, 1972.

Artificial Intelligence Center

Technical Note 64

SRI Project 1530

This lecture is based upon research supported at SRI by the Advanced Research Projects Agency of the Department of Defense, monitored by the U. S. Army Research Office-Durham under Contract DAHC04 72 C 0008.

| Report Documentation Page | | | | | *Form Approved*<br>*OMB No. 0704-0188* |
|---|---|---|---|---|---|
| Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. | | | | | |
| 1. REPORT DATE<br>**FEB 1972** | | 2. REPORT TYPE | | | 3. DATES COVERED<br>**00-02-1972 to 00-02-1972** |
| 4. TITLE AND SUBTITLE<br>**Robot Research at Stanford Research Institute** | | | | | 5a. CONTRACT NUMBER |
| | | | | | 5b. GRANT NUMBER |
| | | | | | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) | | | | | 5d. PROJECT NUMBER |
| | | | | | 5e. TASK NUMBER |
| | | | | | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>**Artificial Intelligence Center,SRI International,333 Ravenswood Avenue,Menlo Park,CA,94025** | | | | | 8. PERFORMING ORGANIZATION REPORT NUMBER |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | | | | | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | | | | | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |
| 12. DISTRIBUTION/AVAILABILITY STATEMENT<br>**Approved for public release; distribution unlimited** | | | | | |
| 13. SUPPLEMENTARY NOTES | | | | | |
| 14. ABSTRACT | | | | | |
| 15. SUBJECT TERMS | | | | | |
| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES<br>**25** | 19a. NAME OF RESPONSIBLE PERSON |
| a. REPORT<br>**unclassified** | b. ABSTRACT<br>**unclassified** | c. THIS PAGE<br>**unclassified** | | | |

**Standard Form 298 (Rev. 8-98)**
Prescribed by ANSI Std Z39-18

## ABSTRACT

This paper begins by reviewing briefly the history of research in pattern recognition, problem solving, and robot systems. Then the evolution of "Shakey," the SRI robot system, is described. The first version of Shakey used a theorem prover for solving problems, a separate complex scene-analysis program, duel symbolic and geometric models of the world, and a simple, error-prone executive system. The second version, recently completed, consists of a more integrated software structure with new mechanisms for planning, learning, and recovering from errors. After discussing some problems for future versions of the Shakey system, the paper concludes with a brief discussion of potential industrial applications of robot research.

## A. Background

People have been fascinated by the idea of robots--intelligent artificial mechanisms--throughout history. According to Jewish folklore, in the 15th Century the chief Rabbi of Prague created the Golem, a man he built out of clay which God helped him imbue with a limited form of life so that it could perform for many years as his servant. I have heard it rumored that many contemporary computer scientists, including John Von Neuman, Norbert Weiner, and Marvin Minsky all claim to be direct descendants of that Rabbi.

In the late 17th Century Baron Von Kempeln toured Europe exhibiting his Chess automaton--a wooden model of a man with a movable arm that would reach out and play chess with all comers. This device attracted wide attention and performed before many of the courts of Europe--until one day the midget inside sneezed during a game.

The first modern robot is probably the system built by Henry Ernst at MIT about ten years ago. Ernst took a remote manipulator designed for manual control and connected it to a small computer. He then developed a programming language for commanding the arm to move about, and attached some sensors--touch and crude photo sensors for the "finger tips"--with which the arm could perceive certain facets of its environment. The most complex task this system could do was grope about a table top and pick up any object that it bumped into.

Current work in the development of robot systems is being pursued in three major laboratories in the United States, one in Great Britain, and several in Japan. The goal of all of these projects seems to be to bring together results from a variety of disciplines and integrate them into a single robot system. This system should then be able to perceive its environment through appropriate sensors, decide what actions to take based on complex problem-solving procedures, and finally actually carry out physical events in the real world by controlling appropriate manipulators or vehicles.

The major theoretical bases for these systems come from two research areas--perceptual pattern-recognition systems and automatic problem-solving systems. In addition, advances in various other technologies have helped form an appropriate platform for this current work. These include electro-mechanical devices, both for industrial materials-handling use and for prosthetic devices; and increases in computer power and advances in computer software such as the development of powerful list-processing languages and multi-access time-shared systems.

## B. Pattern Recognition

A robot must be able to perceive its environment. The most effective sensory dimension--the one that seems capable of capturing the most information in the shortest time--seems to be vision. Therefore the major robot projects have concentrated, until now, on using visual input as the system's principal source of information about the environment. Let us review briefly the history of picture-processing programs.

The first visual perception programs were aimed at recognition of hand-drawn or machine-printed characters. Considerable effort went into the design of trainable machines, such as the Perceptron, which could be taught to discriminate between different classes of inputs. Unfortunately the capabilities of such adaptive systems are severely limited in both practical and theoretical senses, as shown by Minsky and Papert in their recent book.[1*]

[*] References are listed at the end of this paper.

Current interest in visual-perception systems is focussed on _scene analysis_ rather than pattern recognition--that is, programs that extract descriptions from pictures of three-dimensional scenes, rather than programs that classify two-dimensional patterns. One of the first systems of this kind was developed by Roberts in the early 1960s. His system[2] took as its input a photograph of a simple geometric object. By a sequence of local operations on a digitized version of the picture, his program was able to extract from the picture an outline drawing of the object. Higher-level programs could then identify the object type. Thus the scene-analysis problem was factored into two phases: a translation from the raw data into an idealized drawing, and then a recognition or interpretation phase that described the drawing. More recent projects indicate that when the data is derived from typical noisey sources, this sharp division of the problem into two phases is neither possible nor particularly desirable. Still, it is a useful separation for expository purposes.

The first phase of Roberts' process consisted of looking for short line segments--sections of the edges of objects--and then trying to connect them into longer lines to form a complete outline. An alternative approach, implemented by Brice and Fennema,[3] begins with regions rather than lines. The process consists of first dividing the complete picture into elementary regions of uniform intensity and then deciding which regions should be merged together, until the regions approximate the actual surfaces of objects in the scene.

In the mid-1960s Guzman worked on the second phase of the problem: Beginning with a perfect line drawing, how can one divide the scene into its component objects? His system[4] uses an impressive set of heuristics based on the topological structure of edges and surfaces in the scene.

Recent work by Clowes[5] and Huffman[6] approach a similar problem in a much more rigorous, mathematical manner.

The more information a viewer, or a computer program, has about the scene being analyzed, the simpler and more effective the analysis procedure can be. We have become increasingly aware of the necessity for including considerable amounts of "knowledge" in all of our computer programs. Much more information can be extracted from a picture and added to a data base if the data base already contains considerable background information about what it expects to see in the picture. In the extreme, the program may expect to be viewing a picture that contains one of only a small number of possible objects and therefore the recognition process of what particular object is in that scene can consist of a decision tree of simple tests. For example, if a scene is known to contain either a rectangular object or a triangular object, then the scene-analysis program need merely look for pairs of parallel lines in the picture in order to determine what type of object it contains. This was the approach we took at SRI for the first version of the robot system to be discussed below.

## C. Problem Solving

In order to write computer programs that "solve problems" we must develop some clear definition of what we mean by a problem. One approach is to say that a problem is well defined only when we have a clear test for its solution. For example, in Chess the problem is to produce a position in which, according to the rules of the game, the opponent is in Checkmate. In calculus, an integration "problem" is to produce an expression whose derivative is the initially given expression; a proposed solution can be tested easily by the simple algorithmic process of differentiation.

Following this approach, we define a "problem" for a robot system by describing a desired state of the world. The implicit problem for the robot is to transform the initial state of the world into the desired one. Of course, in order to carry out such a transformation the robot must have available and be aware of certain physical capabilities--the legal moves or rules--that it can exercise to change states of the world.

One approach to solving problems was proposed by Newell and his colleagues as a model of human problem-solving behavior.[7] This general problem-solving system contained a problem-independent part consisting of rules for manipulating abstract objects and operators, and a part, dependent upon the problem domain, that defined the nature of the particular objects and operators of interest. At SRI we felt that a more rigorous formal approach might be more effective, easier to apply to new situations, and more logically complete, although it certainly would not be as close a model of human problem-solving behavior.

However, it is certainly not obvious how to encode the necessary information about the world into a formal logical language. Suppose we represent the fact that a robot is at a position P by an atomic formula At(P).* If this is an axiom of logic in the usual sense, then it is an assertion that must be true for all time. What then should we do when the robot rolls to another position? We cannot have axioms arbitrarily changing from true to false or appearing and vanishing.

Green's technique[8] for handling this problem was to add another argument to each predicate that depended upon the state of the world. This argument, then, explicitly showed the state dependence. Thus, At(P,S) means the robot is at position P in state S. When the robot moves to another position, the state is no longer S, so the fact that the robot was at P when the state was S is true for all time.

We may represent actions in this "situation calculus" by functions that map states into new states--the states produced by carrying out the

*We shall assume familiarity with the ideas discussed in the previous lecture, "The Role of Formal Theorem Proving in Artificial Intelligence."

actions. For example, let go(x,y,s) be the state produced after the robot, who happens to be at position x when the state is s, goes to position y. Then, the effect of going from one place to another can be described once and for all by the axiom, $(\forall x)(\forall y)(\forall s)[At(x,s) \supset At(y,go(x,y,s))]$. Plans of action may now be generated by an ordinary theorem prover as follows: The theorem to be proved is simply an assertion that a state having desired properties exists. If that theorem can be proved then the answer-extraction mechanism mentioned in our discussion of predicate-calculus proof procedures will produce a plan for achieving the desired state. For example, suppose we know that the robot is at position A in the initial state of the world $S_o$. Suppose further that we know that there is a window W at position B, that W is initially open, and that the robot may close the window if he is at its position. Figure 1 shows a set of axioms representing all of this information. If we wish the robot to close the window we merely assert the existence of a state in which the window is closed:

$$(\exists s)Closed(W,s).$$

Figure 2 shows the resolution proof that such a state exists. The extracted answer asserts that the window W is closed in the state gotten to by shutting it, after the robot has gone from A to B, starting from state $S_o$.

Axiom 6 in Figure 1 and Figure 2 needs some further explanation. We know that the window is open in state $S_o$. We know further that when the robot goes from A to B the world will be in a new state, namely $S_1 = go(A,B,S_o)$. Now, do we know whether the window is open or closed in state $S_1$? Only implicitly, because we know that going from place to place does not close windows; however, how can the system know that? Only by having an explicit axiom, axiom 6, that asserts that the act of going does not change the property of openness. The need for such property-maintaining axioms is one ramification of the "frame problem"--a perennial bookkeeping problem in

FIGURE 1

<u>Axioms for the Window Problem</u>

| | | |
|---|---|---|
| 1. | $At(A,S_o)$ | The robot is at position A in state $S_o$. |
| 2. | $Open(W,S_o)$ | W is an open window in state $S_o$. |
| 3. | $Place(W,B)$ | W is at position B (independent of states, i.e. its location in the room is not changeable). |
| 4. | $(\forall x)(\forall y)(\forall s)[At(x,s) \supset At(y,go(x,y,s))]$ | The robot will be at whatever position it may "go" to. |
| 5. | $(\forall u)(\forall v)(\forall s)[At(u,s) \wedge Place(v,u) \wedge Open(v,s) \supset Closed(v,shut(v,s))]$ | If the robot is at the same location as a window that is open, then the window will be closed after the "shut" operation is performed. |
| 6. | $(\forall x)(\forall y)(\forall z)(\forall s)[Open(x,s) \supset Open(x,go(y,z,s))]$ | An open window remains open when the robot moves. |

FIGURE 2

How to Shut the Window

| | | |
|---|---|---|
| 1. | $At(A,S_o)$ | Axioms |
| 2. | $Open(W,S_o)$ | |
| 3. | $Place(W,B)$ | |
| 4. | $\sim At(x,s) \quad At(y,go(x,y,s))$ | |
| 5. | $\sim At(u,s) \quad \sim Place(v,u) \quad \sim Open(v,s) \quad Closed(v,shut(v,s))$ | |
| 6. | $\sim Open(x,s) \quad Open(x,go(y,z,s))$ | |
| 7. | $\sim Closed(W,s) \quad Closed(W,s)$ | Negated Theorem |
| 8. | $\sim At(u,s) \quad \sim Place(W,u) \quad \sim Open(W,s) \quad Closed(W,shut(W,s))$ | From 5 & 7 |
| 9. | $\sim At(B,s) \quad \sim Open(W,s) \quad Closed(W,shut(W,s))$ | From 3 & 8 |
| 10. | $\sim At(x,s) \quad \sim Open(W,go(x,B,s)) \quad Closed(W,shut(W,go(x,B,s)))$ | From 4 & 9 |
| 11. | $\sim Open(W,go(A,B,S_o)) \quad Closed(W,shut(W,go(A,B,S_o)))$ | From 1 & 10 |
| 12. | $\sim Open(W,S_o) \quad Closed(W,shut(W,go(A,B,S_o)))$ | From 6 & 11 |
| 13. | $\square \quad Closed(W,shut(W,go(A,B,S_o)))$ | From 2 & 12 |

state-dependent problem solving. This problem has led to our eventual rejection of the situation calculus as a general means for solving robot problems (see the second version of Shakey below).

## D. Control Language

The immediate input to a formal problem-solving system such as the one discussed above must be statements in a formal language such as predicate calculus. However, we recognize that experimentation with the system would be more convenient if the control language were closer to the natural language of the experimenter--ordinary English. Therefore, one of our staff members, Coles, has developed a system that translates a small but interesting subset of ordinary English into predicate calculus.[9] This system contains a transformational grammar that recognizes a variety of syntactic constructs. In addition, certain semantic information is embedded in those rules: namely, what predicate-calculus constructs and relations should be generated as the corresponding "meaning" of the input sentence.

Another project that has just gotten underway at SRI is aimed at a complete speech understanding system. Eventually we hope to be able to direct the robot system in ordinary English through a microphone and, perhaps, carry on a conversation with it about its knowledge and its goals.

## E. Shakey the Robot: First Version

Late in 1969 we completed the first version of a complete robot system.[10] This system was documented in a film entitled "Shakey: A First Generation Robot." It demonstrated a complete, if somewhat crude, combination of the abilities in the component fields discussed above. The control language was limited ordinary English which was translated by the system into first-order predicate calculus. The problem solver, then, was a resolution-type theorem-proving program that used the situation-calculus and answer-construction mechanisms discussed above. The scene-analysis component started with television pictures, applied Roberts' methods to reduce the pictures to line drawings, and then used decision trees to identify significant areas or objects in the scene. Typical problems solved by this system were "GO TO POSITION (X,Y)" and "PUSH THE THREE OBJECTS TOGETHER."

The exercise of assembling Shakey, a complete robot system, was worthwhile for several reasons. First, we discovered major weaknesses with various component systems--the pattern-recognition system, the problem solver, etc. Second, we discovered the immense scale of the computing requirements necessary to assemble such a complex of programs. Our XDS-940 computer (64,000 24-bit words of 1.75 microsecond core memory) was just barely large enough for the simplest problems. The housekeeping problems of fitting all the components into that sized computer and enabling them to communicate with each other were becoming more effort than the basic research program. Third, and perhaps most important, we discovered some research areas that were necessary components of any complete system but that had naturally been overlooked in the course of doing research on the major components separately. These areas were the problem of data representation and the problem of an executive program that could monitor the execution of robot tasks.

### 1. Representations

This first robot system had multiple representations of the world. TV pictures were stored and processed in data arrays. The representation of the floor plan of the room was a grid--a FORTRAN array--that indicated occupied, unoccupied, and unknown regions of the room. This grid was useful for geometric calculations such as route planning but could not contain symbolic descriptive information. Finally, the communications language, theorem-proving, and problem-solving systems contained their own description of the world in the form of symbolic axioms.

These multiple models were awkward to coordinate and update. Changes could not be made simultaneously in all models because of the limitations of space in the computer memory. On the other hand, the bookkeeping required to remember when the models had gotten out of step soon became intractable.

### 2. Executive System

The other difficult area was the design of an executive for the system. In this version of the robot system a trivial program translated the results of a planning process--the sequence of actions needed to accomplish a task--directly into a sequence of calls to subroutines that actually moved the robot about. Nowhere in this sequence was there any opportunity for feedback or other means of checking progress. Execution of a plan was a completely open-loop procedure. Therefore, any long complicated plan had virtually no chance of being successful, even if the plan were essentially correct; the slightest inaccuracy or error anywhere during execution would cause the whole system to fall apart.

## F. Shakey the Robot: Second Version

We have just completed the second complete version of a robot system. The hardware of the robot vehicle itself--a mobile, radio-controlled cart carrying a TV camera and some bumper switches--is virtually unchanged from the first version. However, practically every other component of the entire system is completely new.

The controlling computer system now consists of a PDP-10 with about 200,000 36-bit words of 1 microsecond core memory and a peripheral computer

(PDP-15) for control of the robot, an on-line display, a connection to a nation-wide computer network, and other devices.

The software contains four major levels. At the lowest level we have what we call Low-Level Actions or "LLAs." These are the lowest-level robot-control programs that can be called from user programs that are written in the LISP language, our principal programming tool. The LLAs are programmatic handles on the robot's physical capabilities such as "roll" and "tilt."

So that it can exhibit interesting behavior, our robot system has been equipped with a library of Intermediate-Level Actions or "ILAs." These second-level elements are preprogrammed packages of LLAs embedded in a special interpretative-language framework with various control and error correction features. Each ILA represents built-in expertise in some significant physical capability such as "push" or "go to." The ILAs might be thought of as instinctive abilities of the robot analogous to such built-in complex animal abilities as "walk" or "eat."

The issue of tradeoffs between preprogrammed expertise and general problem-solving capability is a concern in any Artificial Intelligence system. A special-purpose hand-coded program can certainly solve one well-defined problem better than any general problem-solving routine. On the other hand, a single, general problem solver can solve many different problems, although at the present state of our capabilities each of those problems would have to be rather simple and the solutions might not be very efficient. We have drawn the line between expertise and generality at the ILA level. Each ILA may be hand coded and may contain as much tricky cleverness as the programmer wishes to design into it. On the other hand, it should be intended to solve a well-defined and generally useful problem. At higher levels in the system (discussed below), we have general planning mechanisms for assembling arbitrary sequences and combinations of ILAs to perform a wide variety of possible tasks.

The principal sensor of the perceptual system is a TV camera. Programs for processing picture data have been restricted to a few special "vision" routines which are incorporated into the system at either the ILA or LLA level.

Above the ILAs we have the third level, which is concerned with planning the solutions to problems. The basic planning mechanism is a system called STRIPS--a problem solver that uses a combination of heuristic and formal theorem-proving methods.[11] STRIPS constructs sequences of ILAs needed to carry out specified tasks. Such a sequence along with its expected effect can be represented by a triangular table called a macro operation or MACROP.

Finally, the fourth, or top-level of the system, is the executive--the program that actually invokes and monitors execution of the ILAs specified in a MACROP. The current executive program is called PLANEX.[12]

## 1. Shakey's World and Model

Shakey now lives in a laboratory environment consisting of a network of rooms connected by doorways and populated with a variety of boxes of different sizes. The computer representation of this environment is expressed in the form of axioms of first-order predicate calculus (but not including any explicit state argument). It is the responsibility of each LLA that causes changes in the physical world, e.g. "roll," to update the model by making appropriate changes to the axioms, e.g. by modifying the At predicate. Higher-level programs may then always assume that the model, i.e. this set of logical axioms, always describes the present state of the world.

## 2. The STRIPS System

STRIPS is a system for planning sequences of actions that will bring about a desired state of the world. It resembles GPS[7] in the sense that it performs a heuristic-search procedure in order to find a sequence of operators (actions) that will transform the initial object (state of the world) into a desired object (a state having required properties). However, it uses formal theorem-proving techniques to deduce all the logical consequences of anything it knows about any given state. The theorem prover is also used to test the applicability of operators to states, and to determine the principal differences between states, so that STRIPS can select appropriate operators.

Corresponding to each possible physical action of the robot, an operator is available to the STRIPS system. Each operator is described to STRIPS by: a precondition statement, a delete list, and an add list. The precondition statement is a logical theorem that must be provable in the current state of the world in order for the operator to be usable on that state. The delete list and add list contain changes that must be made to the axioms representing the current state in order to transform it into the state that will be produced after the operator is carried out.

For example, one ILA available in the current system is NAVTO--a program that first computes the shortest path between any two points in a room while avoiding known obstacles in that room, and then calls the appropriate LLAs to actually move the robot along that path to its desired target position. The purpose of NAVTO, then, is to enable Shakey to navigate from any point to any other point, in the same room, whose coordinates are specified. The corresponding STRIPS operator, also called NAVTO, has as its only precondition that the robot be in the same room as the desired destination point. The delete list specifies that the current location of the robot, if known, and the possible fact that the robot is next to any known object in the room, must be deleted from the state of the world. The add list specifies that the assertion that the robot is at its destination position must be added. Such descriptions provide STRIPS with just the information it needs to determine the effects of various actions and, therefore, to plan a sequence of actions that will bring about the desired state of the world. The details of how each action is accomplished may be left to the ILAs and LLAs that will actually carry out the actions.

An important feature of our current system is the ability to generalize and save plans produced by STRIPS. For example, suppose the robot is in room A and STRIPS is given the problem, "produce a state of the world in which the robot is in room B." The specific solution that STRIPS would produce, given our present set of ILAs and corresponding operators, is the two-step plan: "Go to the door between room A and room B, and then go through that door into room B." Observe, however, that this plan has extremely limited utility. Suppose sometime in the future we ask the robot to go from room C to room D, or even to go back from room B to room A. Although the particular plan we have just derived would not be useful for these new tasks, clearly all these tasks are equivalent and a single plan should work for any of them. We have recently modified STRIPS so that when asked to solve the problem of getting the robot from room A to room B, STRIPS would actually construct a plan for getting the robot from room x to room y for any two adjacent rooms. That plan would then be stored away as a macro operation, or MACROP, for future reference.

One proposal has been that each MACROP should be stored and available to the system in exactly the same form as all other operators. This is not done, however, because a MACROP is generally considerably richer than a single operator. For example, suppose STRIPS generates a MACROP consisting of a sequence of five operators. As a side effect of the planning process, STRIPS has computed the purpose and effect of each of the five operators separately when they are used in that sequence. This auxiliary information is stored as part of the MACROP (in a particular format called "triangle table"). Now this complete five-operator generalized plan is available for use in future problems. But also, STRIPS can use the auxiliary information to calculate the effects of various <u>subsequences</u> of the five operators and, in fact, such subsequences can be extracted from the plan when appropriate. Therefore, the MACROP is a concise representation for a family of operators each of which can be described and used by STRIPS as needed.

## 3. <u>PLANEX</u>[12]

When STRIPS completes a plan--a new MACROP--for accomplishing a desired task, it turns the MACROP over to PLANEX--the executive for carrying out plans. Now, PLANEX has available to it all the information in a MACROP. This consists not only of the names of the action routines to be called in sequence, but also the expected effects of each routine. PLANEX calls one routine at a time and, after each routine has finished its task, control returns to PLANEX to decide what to do next. PLANEX does not call the next routine in sequence until it has verified both that the previous routine has accomplished its mission and that the next routine is still required. If for any reason some routine does not accomplish its intended purpose or accomplishes something different, PLANEX is free to rearrange the order in which action routines are called in order to achieve the desired result more efficiently. PLANEX can also notice, at an appropriate early point, that the plan is not working and either call STRIPS again or abandon the entire project and ask for help from a human at a Teletype.

### 4. Error Recovery

As just mentioned, PLANEX monitors executions of plans and can attempt to adjust progress and recover from errors at the highest level. In addition, each ILA may have a variety of mechanisms for predicting, testing, and recovering from errors that may be produced while that ILA is operating. Such errors are invisible to higher-level routines and PLANEX, for example, would never be aware of them, provided the ILA that detected the error was capable of correcting it. One example of this type of error occurs when an unknown object is bumped into by the robot. If NAVTO has planned a route avoiding all known obstacles and in the course of traversing that route the robot bumps into a new object, NAVTO can be smart enough to back up, add the newly discovered object to the model, and plan a new route to the goal by going around that object.

Information from the camera can also be very useful for recovering from or avoiding errors. The present system keeps in the robot's model an estimate of the error in the robot's knowledge of its position. Every time an LLA moves the robot, that error estimate is increased. Any LLA that makes use of the information specifying the robot's current position first checks that the error estimate of that position is within an allowable tolerance. If it is not, the visual-processing routine, LANDMARK, is called. LANDMARK is a special-purpose vision routine that takes a picture of the closest "landmark"--that is, an easily recognizable point in the room such as a corner or the edge of a door jamb--and uses measurements made in that picture to update the robot's current position and then to reduce the error estimate of the knowledge of that position.

## G. Shakey the Robot: Future Versions

Having achieved the plateau of a second complete robot system (just described) we are now looking toward the future and listing requirements and research problems for our next few years' efforts.

## 1. Scene Analysis

Our picture-processing programs have evolved during these past five years from complex but largely undirected operations, such as those of Roberts, through very simple but highly directed analysis algorithms, such as the decision trees, to the current system in which the visual-analysis process has been relegated to highly specialized subordinate routines such as LANDMARK. Our future work in perception will proceed in two directions. First, we feel that the use of light intensity obtained from a TV camera as our only source of data is restrictive. We are beginning to experiment with the use of depth and color as additional sensory dimensions. Second, we feel that the analysis of sensory data is a problem-solving task and, therefore, the sensory-perception and the problem-solving components of a robot system must be better integrated. We are just beginning to explore how this can be done.

## 2. Problem Solving

We are pleased with the success of STRIPS and PLANEX for the classes of problems that we have considered thus far. However, the nature of the problems we have considered and the nature of the experimental domain have all been rather restrictive. Many problems are difficult to pose in the form of state descriptions. For example, "go down the hall and take a picture through every open doorway" cannot be defined by a single state predicate, unless we use some awkward and unsatisfying coding tricks. Therefore, we must define richer languages for stating problems--and this means reopening the question of what is a problem, as opposed to an algorithm for solving it.

A plan, as we now use it, is always a linear sequence of operator calls. For solving problems like "go down the hall ..." we should be able to construct plans with loops. How difficult will it be to generate plans that contain such conventional programming features as loops and conditional branches? A plan of activity for a robot resembles, in many ways, a program of instructions for a computer. One area we wish to explore is the applicability of the problem-solving methods we have been studying to the automatic construction of computer programs.

Thus far we have been working in a static experimental environment in which a single robot vehicle is the only active element. We plan very soon to begin studying problems involving dynamic environments--that is, environments that may change from time to time without the knowledge of the robot and may even contain moving objects. We will also be interested in the problem of two or more robots cooperating with each other while under the control of a single computer or, alternatively, while being operated independently so that they can only perceive each other's activities through appropriate sensors.

## 3. Implementation Tools

All of our work thus far has been developed in a time-shared computing environment using a combination of LISP and FORTRAN programming languages. A new generation of programming languages for implementing problem-solving systems is now becoming available. Languages such as PLANNER[13] will allow the system designer to think in terms of parallel processes, cooperating coroutines, "demons" monitoring the progress of various programs, and theorem-proving and heuristic-search procedures implicitly built into the language. The usefulness of these new kinds of tools for designing and building robot systems remains to be explored.

## H. Applications for Industry

The work discussed above has been essentially long-range, theoretical research--fundamental studies into the nature of intelligence and perception and how they may be emulated by automatic systems. We have gained considerable knowledge of problem-solving and perceptual systems and have just begun to explore how these systems can be coupled to physical devices in the real world.

On the other hand, industry has recently been developing a variety of sophisticated physical devices. These devices include numerically controlled machines, "teleoperators" or remotely controlled slave devices, and programmed manipulators. These devices are all striking in that they have considerable dexterity and strength but virtually no perceptual or problem-solving ability.

The time appears ripe to marry these two lines of development. We believe that by borrowing even the most trivial aspects of perceptual and decision-making capabilities from the research laboratories and integrating them with existing systems for visual inspection and materials handling one could very quickly produce automated systems of much greater value than any in use today. At SRI we have recently begun a development program to explore such possibilities.

## REFERENCES

1. M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry (The MIT Press, Cambridge, Mass., 1969).

2. L. G. Roberts, "Machine Perception of Three-Dimensional Solids," in Optical and Electro-Optical Information Processing, pp. 159-197, J. T. Tippett et al., Eds. (The MIT Press, Cambridge, Mass., 1965).

3. C. R. Brice and C. L. Fennema, "Scene Analysis Using Regions," Artificial Intelligence, Vol. 1, No. 3, pp. 205-226 (American Elsevier Publishing Company, New York, NY, 1970).

4. A. Guzman, "Decomposition of a Visual Scene into Three-Dimensional Bodies," Proceedings FJCC, pp. 291-304 (December 1968).

5. M. B. Clowes, "On Seeing Things," Artificial Intelligence, Vol. 2, No. 1, pp. 79-116 (North-Holland Publishing Company, Amsterdam, The Netherlands, 1971).

6. D. A. Huffman, "Impossible Objects as Nonsense Sentences," in Machine Intelligence 6, pp. 295-323, B. Meltzer and D. Michie, Eds. (Edinburgh University Press, Edinburgh, Scotland, 1971).

7. G. Ernst and A. Newell, GPS: A Case Study in Generality and Problem Solving (ACM Monograph Series, Academic Press, New York, NY, 1969).

8. C. Green, "Application of Theorem Proving to Problem Solving," Proceedings IJCAI (The Mitre Corporation, Bedford, Mass., 1969).

9. L. S. Coles, "The Application of Theorem Proving to Information Retrieval," Proceedings Fifth Hawaii International Conference on System Sciences, Honolulu, Hawaii, January 1972.

10. C. A. Rosen, "An Experimental Mobile Automaton," Proceedings American Nuclear Society Eighteenth Conference on Remote Systems Technology, Washington, DC, November 1970.

11. R. E. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2, Nos. 3/4, pp. 189-208 (1971).

12. R. E. Fikes, "Monitored Execution of Robot Plans Produced by STRIPS," Proceedings IFIP Congress '71, Ljubljana, Yugoslavia, August 1971.

13. C. Hewitt, "PLANNER: Language for Proving Theorems in Robots," Proceedings IJCAI (The Mitre Corporation, Bedford, Mass., 1969).