A MOBIUS AUTOMATON: AN APPLICATION OF ARTIFICIAL INTELLIGENCE TECHNIQUES

by

Nils J. Nilsson
Stanford Research Institute
Menlo Park, California

Summary

A research project applying artificial intelligence techniques to the development of integrated robot systems Is described. The experimental facility consists of an SDS-940 computer and associated programs controlling a wheeled vehicle that carries a TV camera and other sensors. The primary emphasis is on the development of a system of programs for processing sensory data from the vehicle, for storing relevant information about the environment, and for planning the sequence of motor actions necessary to accomplish tasks in the environment. A typical task performed by our present system requires the robot vehicle to rearrange (by pushing) simple objects in its environment .

A novel feature of our approach is the use of a formal theorem-proving system to plan the execution of high-level functions as a sequence of other, perhaps lower-level, functions. The execution of these, in turn, requires additional planning at lower levels. The main theme of the research is the integration of the necessary planning systems, models of the world, and sensory processing systems into an efiicient whole capable of performing a wide range of tasks in a real environment.

Key Words

Robot
Robot System
Visual Processing
Problem-Solving
Question-Answering
Theorem-Proving
Models of the World
Planning
Scene Analysis
Mobile Automaton

Acknowledgment

At least two dozen persons at the Stanford Research Institute have made substantial contributions to the project that tne author has the good fortune to describe in this paper. All of us express our appreciation to the Rome Air Development Center and the Advanced Research Projects Agency, who supported this research under Contract No. F 30602-69-C-0056.

I Introduction

At the Stanford Research Institute we are implementing a facility for the experimental study of robot systems. The facility consists of a time-shared SDS-940 computer, several core-loads of programs, a robot vehicle, and special interface equipment.

Several earlier reports[1] and papers[2-4] describing the project have been written; in this paper we shall describe its status as of early 1969 and discuss some of our future plans.

The robot vehicle itself is shown in Fig. 1. It is propelled by two stepping motors independently driving a wheel on either side of the vehicle. It carries a vidicon television camera and optical range-finder in a movable "head." Control logic on board the vehicle routes commands from the computer to the appropriate action sites on the vehicle. In addition to the drive motors, there are motors to control the camera focus and iris settings and the tilt angle of the head. (A motor to pan the head is not yet used by present programs.) Other computer commands arm or disarm interrupt logic, control power switches, and request readings of the status of various registers on the vehicle. Besides the television camera and range-finder sensors, several "cat-whisker" touch-sensors are attached to the vehicle's perimeter. These touch sensors enable the vehicle to know when it bumps into something. Commands from the SDS-940 computer to the vehicle and information from the vehicle to the computer are sent over two special radio links, one for narrow-band telemetering and one for transmission of the TV video from the vehicle to the computer.

The purpose of our robot research at SRI is to study processes for the real-time control of a robot system that interacts with a complex environment. We want the vehicle to be able to perform various tasks that require it to move about in its environment or to rearrange objects. In order to accomplish a wide variety of tasks rather than a few specific ones, a robot system must have very general methods. What is required is the integration in one system of many of the abilities that are usually found separately in individual Artificial Intelligence programs.

We can group most of the needed abilities into three broad classes: (1) problem-solving, (2) modelling, and (3) perception:

(1) Problem-Solving

A robot system accomplishes the tasks given it by performing a sequence of primitive actions, such as wheel motions and camera readings. For efficiency, a task should first be analyzed into a sequence of primitive actions calculated to have the desired effect. This process of task analysis is often called planning, because it is accomplished before the robot begins to act. Obviously, in order to plan, a robot system must "know" about the effects of its actions.

(2) Modelling

A body of knowledge about the effects of actions is a type of model of the world. A robot problem-solving system uses the information stored in the model to calculate what sequence of actions will cause the world to be in a desired state. As the world changes, either by the robot's own actions or for other reasons, the model must be updated to record these changes. Also, new information learned about the world should be added to the model.

(3) Perception

Sensors are necessary to give a robot system new information about the world. By far the most important sensory system is vision, since it allows direct perception of a good sized piece of the world beyond the range of touch. Since we assume that a robot system will not always have stored in its model every detail of the exact configuration of its world and thus cannot know precisely the effects of its every action, it also needs sensors with which to check predicted consequences against reality as it executes Its plans.

The integration of such abilities into a smoothly-running, efficient system presents both important conceptual problems and serious practical challenges. For example, it would be infeasible for a single problem-solving system (using a single model) to attempt to calculate the long chains of primitive actions needed to perform lengthy tasks. A way around this difficulty is to program a number of coordinating "action-units," each with its own problem-solving system and model, and each responsible for planning and executing a specialized function. In planning how to perform its particular function, each action-unit knows the effects of executing functions handled by various of the other action-units. With this knowledge it composes its plan as a sequence of other functions (with the appropriate arguments) and leaves the planning required for each of these functions up to the action-units responsible for executing them at the time they are to be executed.

Such a system of interdependent action-units implies certain additional problems involving communication of information and transfer of control between units. When such a system is Implemented on a serial computer with limited core memory, obvious practical difficulties arise connected with swapping program segments in and out of core and handling interrupts in real time. The coordinated action-unit scheme serves as a useful guide in explaining the operation of our system, even though practical necessities have dictated occasional deviations from this scheme in our implementation. In the next section we shall discuss the problem-solving processes and models associated with some specific functions of the present SRI robot system.

II SOME SPECIFIC FUNCTIONS OF THE ROBOT SYSTEM AND THEIR ASSOCIATED PROBLEM-SOLVING PROCESSES AND MODELS

A. Low Level Functions

The robot system is capable of executing a number of functions that vary in complexity from the simple ability to turn the drive wheels a certain number of steps to the ability to collect a number of boxes by pushing them to a common area of the room. The organization of these functional action-units is not strictly hierarchical, although for descriptive convenience we will divide them into two classes: low level and high level functions.

Of the functions that we shall mention here, the simplest are certain primitive assembly language routines for moving the wheels, tilting the head, reading a TV picture, and so on. Two examples of these are MOVE and TURN; MOVE causes the vehicle to roll in a straight line by turning both drive wheels in unison, and TURN causes the vehicle to rotate about its center by turning the drive wheels in opposite directions. The arguments of MOVE and TURN are the number of steps that the drive wheels are to turn (each step resulting in a vehicle motion of 1/32 inch) and "status" arguments that allow queries to be made about whether or not the function has been completed.*

Once begun, the execution of any function proceeds either until it Is completed in its normal manner or until it is halted by one of a number of "abnormal" circumstances, such as the vehicle bumping into unexpected objects, overload conditions, resource exhaustion, and so on. Under ordinary operation, if execution of MOVE results in a bump, mollon is stopped automatically by a special mechanism on the vehicle. This mechanism can be overridden by a special instruction from the computer, however, to enable the robot to push ob Jects.

The problem-solving systems for MOVE and TURN are trivial; they need only to calculate what signals shall be sent to registers associated with the motors in order to complete the desired number of steps.

At a level just above MOVE and TURN is a function whose execution causes the vehicle to travel directly to a point specified by a pair of (x,y) coordinates. This function is implemented in the FORTRAN routine LEG. The model used by LEG contains information about the robot's present (x,y) location and heading relative to a given coordinate system and information about how far the vehicle travels for each step applied to the stepping motors. This information is stored along with some other special constants in a structure called the PARAMETER MODEL. Thus for a given (x,y)

* Our implementation allows a program calling routines like MOVE or TURN to run in parallel with the motor functions they initiate.

destination as an argument of LEG, LEG's problem-solving system calculates appropriate arguments for a TURN and MOVE sequence and then executes this sequence. Predicted changes in the robot's location and heading caused by execution of MOVE and TURN are used to update the PARAMETER MODEL.

Ascending one more level in our system, we encounter a group of FORTRAN "two-letter" routines whose execution can be initiated from the teletype. Our action-unit system ceases to be strictly hierarchical at this point, since some of the two-letter commands can cause others to be executed.

One of these two-letter commands, EX, takes as an argument a sequence of (x,y) coordinate positions. Execution of EX causes the robot to travel from its present position directly to the first point in the sequence, thence directly to the second, and so on until the robot reaches the last point in the sequence. The problem-solving system for EX simply needs to know the effect caused by execution of a LEG program and composes a chain of LEG routines, each with arguments provided by the successive points specified in the sequence of points. Under ordinary operation, if one of these LEG routines is halted due to a bump, EX backs the vehicle up slightly and then halts. A special feature of our implementation is the ability to arm and service interrupts (such as caused by bumps) at the FORTRAN programming level.

Another two-letter command, PI, causes a picture to be read after the TV camera has been aimed at a specified position on the floor. The problem-solving system for PI thus calculates the appropriate arguments for a TURN routine and a head-tilting routine; PI then causes these to be executed, reads in a picture from the TV camera, and performs processing necessary to extract information about empty areas on the floor. (Details of the picture processing programs of the robot system are described in Sec. III below.)

The ability to travel by the shortest route to a specified goal position along a path calculated to avoid bumping into obstacles is provided by the two-letter command TE. Execution of TE involves the calculation of an appropriate sequence of points for EX and the execution of EX. This appropriate sequence is calculated by a special problem-solving system embodied in the two-letter command PL.

The source of information about the world used by PL is a planar map of the room called the GRID MODEL. The GRID MODEL is a hierarchically organized system of four-by-four grid cells. Initially the "whole world" is represented by a four-by-four array of cells. A given cell can be either empty (of obstacles), full, partially full, or unknown. Each partially full cell is further subdivided into a four-by-four array of cells, and so on, until all partially full cells represent areas of some suitably small size. (Our present system splits cells down to a depth of three levels, representing a smallest area of about 12 Inches.)

Special "model maintenance" programs insure that the GRID MODEL is automatically updated by information about empty and full floor areas gained by either successful execution or interruption of MOVE commands.

The PL program first uses the GRID MODEL to compute a network or graph of "nodes." The nodes correspond to points in the room opposite corners of obstacles; the shortest path to a goal point will then pass through a sequence of a subset of these nodes. In Fig. 2 we show a complete GRID MODEL of a room containing three objects. The robot's position, marked "R," and the goal position, marked "G," together with the nodes A,B,C,D, E,F,H,I,J and K are shown overlaid on the GRID MODEL. The program PL then determines that the shortest path is the sequence of points, R,F,I, and G by employing an optimal graph-searching algorithm developed by Hart, et al.5

If the GRID MODEL map of the world contains unknown space, PL must decide whether or not to treat this unknown space as full or empty. Currently, PL multiplies the length of any segment of the route through unknown space by a parameter k. Thus if k=l, unknown space is treated as empty; values of k greater than unity cause routes through known empty space to be preferred to possibly shorter routes through unknown space.

Execution of TE is accomplished by first reading and processing a picture (using PI with the camera aimed at the goal position) and taking a range-finder reading. The information about full and empty floor areas thus gained is added to the GRID MODEL. A route based on the updated GRID MODEL is then planned using PL, and then EX is executed using the arguments calculated by PL. If the EX called by TE is halted by a bump, a procedure attempts to manuever around the interfering obstacle, and then TE is called to start over again. Thus, vision is used only at the beginning of a journey and when unexpected bumps occur along the journey.

Although our present robot system does not have manipulators with which to pick up objects, it can move objects by pushing them. The fundamental ability to push objects from one place to another is programmed into another two-letter FORTRAN routine, called PU. Execution of PU causes the robot to push an object from one named position along a straight line path to another named position. The program PU takes five arguments: the (x,y) coordinates of the object to be pushed, the "size" or maximum extent of the object about its center of gravity, and the (x,y) coordinates of the spot to which the object is to be pushed. The problem-solving system for PU assembles an EX, a TURN, and two MOVE commands into a sequence whose execution will accomplish the desired push. First a location from which the robot must begin pushing the object is computed. Then PL is used to plan a route to this goal location. The sequence of points along the route serves as the argument for EX that is then executed. (Should EX be stopped by a bump, PU is started over again.) Next, PU's

problem-solving system (using the PARAMETER model) calculates an argument for TURN that will point the robot in the direction that the object is to be pushed. A large argument is provided for the first MOVE command so that when it is executed, it will bump into the object to be pushed and automatically halt. After the bump and halt, the automatic stopping mechanism on the vehicle is overridden and the next MOVE command is executed with an argument calculated to push the object the desired distance.

B. Higher Level Functions

As we ascend to higher level functions, the required problem-solving processes must be more powerful and general. We want our robot system to have the ability to perform tasks possibly requiring quite complex logical deductions. What is needed for this type of problem-solving is a general language in which to state problems and a powerful search strategy with which to find solutions. We have chosen the language of first-order predicate calculus in which to state high level problems for the robot. These problems are then solved by an adaptation of a "Question Answering System" QA-3, based on "resolution" theorem-proving methods.[6-9]

As an example of a high level problem for the robot, consider the task of moving (by pushing) three objects to a common place. This task is an example of one that has been executed by our present system. If the objects to be pushed are, say, OBI, 062, and 0B3, then the problem of moving them to a common place can be stated as a "conjecture" for QA-3:

C7p,s){POSITION (0BI,p,s)
A POSITION (OB2,p,s)
A POSITION (0B3,p,s)}

(That is, "There exists a situation s and a place p, such that OBI, 0B2, and 0B3 are all at place p in situation s.") The task for QA-3 is to "prove" that this conjecture follows from "axioms" that describe the present position of objects and the effects of certain actions.

Our formulation of these problems for the theorem-prover involves specifying the effects of actions in terms of functions that map situations into new situations. For example, the function PUSH (x,p,s) maps the situation s into the situation resulting by pushing object x into place p. Thus two axioms needed by QA-3 to solve the pushing problem are:

(Vx,p,s){POSITION (x,p, PUSH (x,p,s))}

and

(Vx,y,p,q,s){POSITION (x,p,s) A ~ SAME (x,y)
*=> POSITION (x,p,PUSH (y,q,s))} .

The first of these axioms states that if in an arbitrary situation s, an arbitrary object x is pushed to an arbitrary place p, then a new situation, PUSH (x,p,B), will result in which the object x will be at position p. The second axiom Btates that any object will stay in its old place in the new situation resulting by pushing a different object.

In addition to the two axioms Just mentioned, we would have others describing the present positions of objects. For example, if OBI is at coordinate position (3,5) in the present situation, we would have:

POSITION (OBI, (3,5), PRESENT) .

(This information is provided automatically by routines that scan the GRID MODEL, giving names to clusters of full cells and noting the locations of these clusters.)

In proving the truth of the conjecture, the theorem-prover used by QA-3 also produces the place p and situation s that exist. That is, QA-3 determines that the desired situation s is:

s = PUSH (0B3, (3,5), PUSH (0B2, (3,5), PRESENT)).

All of the information about the world used by QA-3 in solving this problem is stored in the form of axioms in a structure called the AXIOM MODEL. In general, the AXIOM MODEL will contain a large number of facts, more than are necessary for any given deduction.

Another LISP program examines the composition of functions calculated by QA-3 and determines those lower level FORTRAN two-letter commands needed to accomplish each of them. In our present example, a sequence of PU commands would be assembled. In order to calculate the appropriate arguments for each PU, QA-3 is called again, this time to prove conjectures of the form:

(^p,w){POSITION (OB2,p,PRESENT) A SIZE (OB2,w)) .

Again the proof produces the p and w that exist, thus providing the necessary position and size arguments for PU. (Size information is also automatically entered into the AXIOM MODEL by routines that scan the GRID MODEL.)

In transferring control between LISP and FORTRAN (and also between separate large FORTRAN segments), use Is made of a special miniature monitor system called the VALET. The VALET handles the process of dismissing program segments and starting up new ones using auxiliary drum storage for transferring information between programs.

The QA-3 theorem-proving system allows us to pose quite general problems to the robot system, but further research is needed on adapting theorem-proving techniques to robot problem-solving in order to Increase efficiency.* The generality of

* We can easily propose less fortunate axlomatizatlons for the "collecting objects task" that would prevent QA-3 from being able to solve it.

theorem-proving techniques tempts us to use a single theorem-prover (and axiom set) as a problem-solver (and model) for all high level robot abilities. We might conclude, however, that efficient operation requires a number of coordinating action-unit structures, each having its own specialized theorem-prover and axiom set and each responsible for relatively narrow classes of functions.

Another LISP program enables commands stated in simple English to be executed.[10,*11] It also accepts simple English statements about the environment and translates them into predicate calculus statements to be stored as axioms. English commands are ordinarily translated into predicate calculus conjectures for QA-3 to solve by producing an appropriate sequence of subordinate functions. For some simple commands, the theorem-prover is bypassed and lower level routines such as PU, TE, etc., are called directly.

The English program also accepts simple English questions that require no robot actions. For these, it uses QA-3 to discover the answer, and then it delivers this answer in English via the teletypewriter. (Task execution can also be reported by an appropriate English output.)

III VISUAL PERCEPTION

Vision is potentially the most effective means for the robot system to obtain information about its world. The robot lives in a rather antiseptic but nevertheless real world of simple objects—boxes, wedges, walls, doorways, etc. Its visual system extracts information about that world from a conventional TV picture. A complete scene analysis would produce a description of the visual scene, including the location and identification of all visible objects. Currently, we have two separate operating vision programs. One of these produces line drawings, and has been used for some time to identify empty floor space, regions on the floor into which the robot is free to move. The other, which is more recent, locates and identifies the major, non-overlapping objects. In this section we shall give brief descriptions of how these programs operate.

A. Line Drawing Program

The line drawing program produces a line drawing representation of a scene by a series of essentially local operations on a TV picture.* Fig. 3a shows a typical digitized picture, which is stored in the computer as a 120 x 120 array of 4-bit (16-level) intensity values. The scene shown is fairly typical, and includes some of the problems of mild shadows and reflections, some faint edges, and objects not completely in the field of view.

* Most of these operations were adaptations of earlier work by Roberts.[12] Details of our procedures, together with a description of special hardware for doing them efficiently, are given in Refs. 1 and 4.

The first of the local operations is a digital differentiation used to find points where there are significant changes in light intensity. These changes usually occur at or near the boundaries of objects, as can be seen from Fig. 3b. The next step is to determine the local direction of these boundaries. This is done by systematically placing small masks over the differentiated picture, and looking for places where the masks line up well with the gradient. Fig. 3c shows the locations and orientations of masks that responded strongly.

The next step is to fit these short line segments with longer straight lines. This is done by first grouping the short line segments, and then fitting a single straight line to all of the segments in a group. Grouping is a systematic procedure in which short segments are linked if they are sufficiently close in location and orientation. Fig. 3d shows the results of fitting longer lines to the segments in each group. The final step is to Join the endpoints of these long lines to produce a connected line drawing. This is done by considering the endpoints one at a time and creating candidate connections—straight connections to neighboring endpoints, extrapolations to points of intersection, extrapolations to T-junctions, etc. The candidate that best fits the corresponding part of the derivative picture is the one selected. The final line drawing produced by this procedure is shown in Fig. 3e.

While the line drawing preserves much of the information in the quantized picture in a compact form, it often contains flaws due to missing or extra line segments that complicate its analysis. Currently, the only information we extract from the line drawing is a map of the open floor space. A program called floor boundary first finds a path along those line segments that bound the floor space in the picture. These lines are typically the places where the walls or objects meet the floor, or where sides of objects obscure our view of the floor. Fig. 3f shows the floor boundary extracted from the line drawing.

Now corresponding to any point in the picture is a ray going from the lens center of the camera through the picture point and out into space. This ray is the locus of all points that can produce the given picture point. If we follow the rays going through points on the floor boundary to the points at which they pierce the floor, we obtain an irregular polygon on the floor that bounds space known to be empty. In this way the line drawing is used to identify empty floor space, and the vision system enters information about open area into the GRID MODEL.

B. Object Identification Program

Were the line drawing program able to produce a perfect line drawing, the analysis needed to locate and identify objects in the scene would be relatively straightforward. However, the line drawing often contains flaws that seriously complicate its analysis. Some of these flaws could be corrected by more elaborate local processing.

However, there is a limit to how well local processing can perform, and when significant edges cannot be told from insignificant edges *on* the basis of local criteria, the goal of producing a perfect line drawing in this way must be abandoned.

The object identification program locates and identifies non-overlapping objects by gathering and interpreting evidence supplied by local operators. The program consists of two parts: a repertoire of local operators and an executive. The local operators process the gradient picture to perform tests such as deciding whether or not there is a line between two points, or finding all lines leaving a given point. Each operator returns not a single answer, but a set of possible answers with associated confidences (ideally, probabilities) that each answer is in fact correct. The executive explores the scene by calling local operators and evaluating the results in the light of both prior test results and built-in knowledge of the world.

The executive program is organized as a decision tree. Each node in the tree specifies that a particular test is to be performed. The branches leaving a node correspond to the possible test outcomes, and since each outcome has an associated confidence, these confidences are attached to the branches.

A given node in the tree can be thought of as representing a hypothesis about the contents of the scene. This hypothesis is simply that the scene is partially described by the test results specified by the path from the start node to the given node. The hypothesis is given a confidence by combining the confidences of these test results. The test called for at the given node is designed to provide an answer that will tend to confirm or infirm the hypothesis.

An analysis of a scene proceeds as a search of the decision tree described. At any stage in the search we have a partially expanded tree corresponding to the tests already performed. The nodes at the tips of this partial tree, which we shall call open nodes, present us with a choice of possible next tests (or, alternatively, hypotheses to be further investigated). The open node with highest associated confidence is selected for expansion, i.e., the test called for by that particular node is performed. The search proceeds until the open node with highest confidence is a terminal node of the tree. A terminal node typically represents a complete description of at least a portion of the scene, and hence constitutes at least a partial "answer," (Certain terminal nodes correspond to impossible physical situations; in this event, the search Is resumed at the next most confident node.) After returning a partial answer the portion of the scene containing the object found is deleted and the analysis begins again. Thus the tree search is iterated until the scene contains no further objects.

The decision tree itself embodies the strategy for searching the scene. The basic ideas behind this strategy are simple, and will be illustrated by following the operation of the program on the scene of Fig. 3a. The first operation called for is a search for vertical lines, since these are usually both reliably detectable and significant. Fig. 4a shows that the appropriate operator found three vertical lines, which happened to rank in confidence as numbered.

Starting with the highest confidence line and checking to see that its lower endpoint was within the picture, the program next looked for other lines leaving that endpoint. A failure here would have led to the conclusion that something was strange, and therefore to a transfer to the next most confident node in the tree. However, a "spur" was detected, as shown in Fig. 4b. Hypotheses that the lower endpoint was connected to the lower endpoints of other verticals were rejected because the direction of the spur was not correct. Thus, at this point attention was shifted to the top of the vertical. A spur was found there, as expected, and that spur was followed to its endpoint as shown in Fig. 4c. The fact that its endpoint was on the picture, coupled with the fact that the program had failed in its attempt to connect the vertical to other verticals, provided strong evidence that a wedge had been found. A further check of the angle at the top of the vertical confirmed the wedge hypothesis, and the same calculations used in the floor boundary program were used to locate the lower vertices.

At this point one object had been found and identified, and a search for other objects began. On the second iteration, the remaining verticals were successfully Joined at their lower endpoints, as shown in Fig. 4d, and various spurs were found, as shown in Fig. 4e. A similar attempt to spot a wedge failed to produce strong evidence, as shown in Fig. 4f, and the final output indicated the existence and location of an object partly out of view without specifying what it was. At this point there were no more vertical lines, and the analysis was completed.

The object identification program is capable of locating and often identifying non-overlapping objects on the basis of partial information. There are a number of obvious ways in which it can be improved and extended, and further research will be devoted to these tasks. However, even as it stands it can provide the robot with much valuable information about the robot's world.

IV CONCLUSIONS

There are several key questions that our work has helped to put into focus. Given that a robot system will involve the successful integration of problem-solving, modelling, and perceptual abilities, there are many research questions concerning each of these. Let us discuss each in turn.

A. Problem-Solving

Our somewhat hierarchical organization of problem-solvers and models seems a natural, even if ad hoc, solution to organizing complex behavior. Are there alternatives? Will the use of theorem-proving techniques provide enough generality to

permit a single general-purpose problem-solver, or will several "specialist" theorem-provers be needed to gain the required efficiency?

Other questions concern the use of theorem-proving methods for problem-solving. How do they compare with the "production methods" as used by the General Problem Solver (GPS) or with the procedural language approach as developed by Fikes?[13] Perhaps some combination of all of these will prove superior to any of them; perhaps more experience will show that they are only superficially different.

Another question is: To what level of detail should behavioral plans be made before part of the plan is executed and the results checked against perceptual information? Although this question will not have a single answer, we need to know upon what factors the answer depends.

Our problem-solving research will also be directed at methods for organizing even more complex robot behavior. We hope eventually to be able to design robot systems capable of performing complex assembly tasks requiring the intelligent use of tools and other materials.

B. Modelling

Several questions about models can be posed: Even if we continue to use a number of problem-solvers, must each have its own model? To what extent can the same model serve several problem-solvers? When a perceptual system discovers new information about the world, should it be entered directly into all models concerned? In what form should information be stored in the various models? Should provisions be made for forgetting old information? Can a robot system be given a simple model of itB own problem-solving abilities? Ensuing research and experience with our present system should help us with these questions.

C. Visual Perception

The immediate vision problems involve including more tests in the object identification program to complete unfinished analysis, and removing the restriction to non-overlapping objects. Beyond these improvements there are still longer range problems to be solved. The scene analysis programs implicitly store information about the world in their structure. Changes in the robot's world can require extensive changes to the whole program. What program organization would minimize these problems? How can the scene analysis program interrogate and use facts stored in the model to advantage? Since "facts" obtained from either the model or the subroutines are subject to error, it is natural to accompany them by a confidence measure. How should these confidences be computed and how should they be combined, since, loosely speaking, we operate under conditions of strong statistical dependence? How can we augment the current repertoire of subroutines with others to make use of such properties as color, texture and range? Future vision research will be devoted to answering questions such as these.

The main theme of the project has been, and will continue to be, the problem of system integration. In studying robot systems that interact with the real world, it seems extremely important to build and program a real system and to provide it with a real environment. Whereas much can be learned by simulating certain of the necessary functions (we use this strategy regularly), many important issues are likely not to be anticipated at all in simulations. Thus questions regarding, say, the feasibility of a system of interacting action-units for controlling a real robot can only be confronted by actually attempting to control a real robot with such a system. Questions regarding the suitability of candidate visual processing schemes can most realistically be answered by experiments with a system that needs to "see" the real world. Theorem-proving techniques seem adequate for solving many "toy" problems; will the full generality of this approach really be exploitable for directing the automatic control of mechanical equipment in real-time?

The questions that we have posed in this section are among those that must be answered in order to develop useful and versatile robot systems. Experimenting with a facility such as we have described appears to be the best way to elicit the proper questions and to work toward their answers.

REFERENCES

1. N. Nilsson, et al, "Application of Intelligent Automata to Reconnaissance," Contract AF30(602)-4147, SRI Project 5953, Stanford Research Institute, Menlo Park, California (four Interim Reports and one Final Report dated December 1968).

2. C. A. Rosen and N. J. Nilsson, "An Intelligent Automaton," IEEE International Convention Record, Part 9 (1967).

3. B. Raphael, "Programming a Robot," Proc. IFIP Congress 68, Edinburgh, Scotland (August 1968).

4. G. E. Forsen, "Processing Visual Data with an Automaton Eye," in Pictorial Pattern Recognition (Thompson Book Company, Washington, D.C., 1968).

5. P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Trans, on Systems Science and Cybernetics, Vol. SSC-4, No. 2, pp. 100-107, (July 1968).

6. C. Green and B. Raphael, "The Use of Theorem-Proving Techniques in Question-Answering Systems," Proc. 1968 ACM Conference, Las Vegas, Nevada (August 1968).

7. B. Raphael, "Research on Intelligent Question-Answering Systems," Final Report, Contract AF 19(628)-5919, SRI Project 6001, Stanford Research Institute, Menlo Park, California (May 1968).

8. C. Green, "Theorem-Proving by Resolution as a Basis for Question-Answering Systems," Machine Intelligence 4, B. Meltzer and D. Michie, Eds. (Edinburgh University Press, Edinburgh, Scotland; to appear 1969).

9. C. Green, "Applications of Theorem-Proving to Problem-Solving," Proc. of the International Joint Conference on Artificial Intelligence, Washington, D.C., (May 1969).

10. L. S. Coles, "An On-Line Question-Answering System with Natural Language and Pictorial Input," Proc. 1968 ACM Conference, LAS Vegas, Nevada (August 1968).

11. L. S. Coles, "Talking with a Robot in English," Proc. of the International Joint Conference on Artificial Intelligence, Washington, D.C., (May 1969).

12. L. G. Roberts, "Machine Perception of Three-Dimensional Solids," Optical and Electro-Optical Information Processing (MIT Press, Cambridge, Massachusetts, 1965).

13. R. Fikes, "A 9tudy in Heuristic Problem-Solving: Problems Stated as Procedure," Proc. of Fourth Systems Symposium, held at Case Western Reserve University, Cleveland, Ohio, November 1968 (to be published).

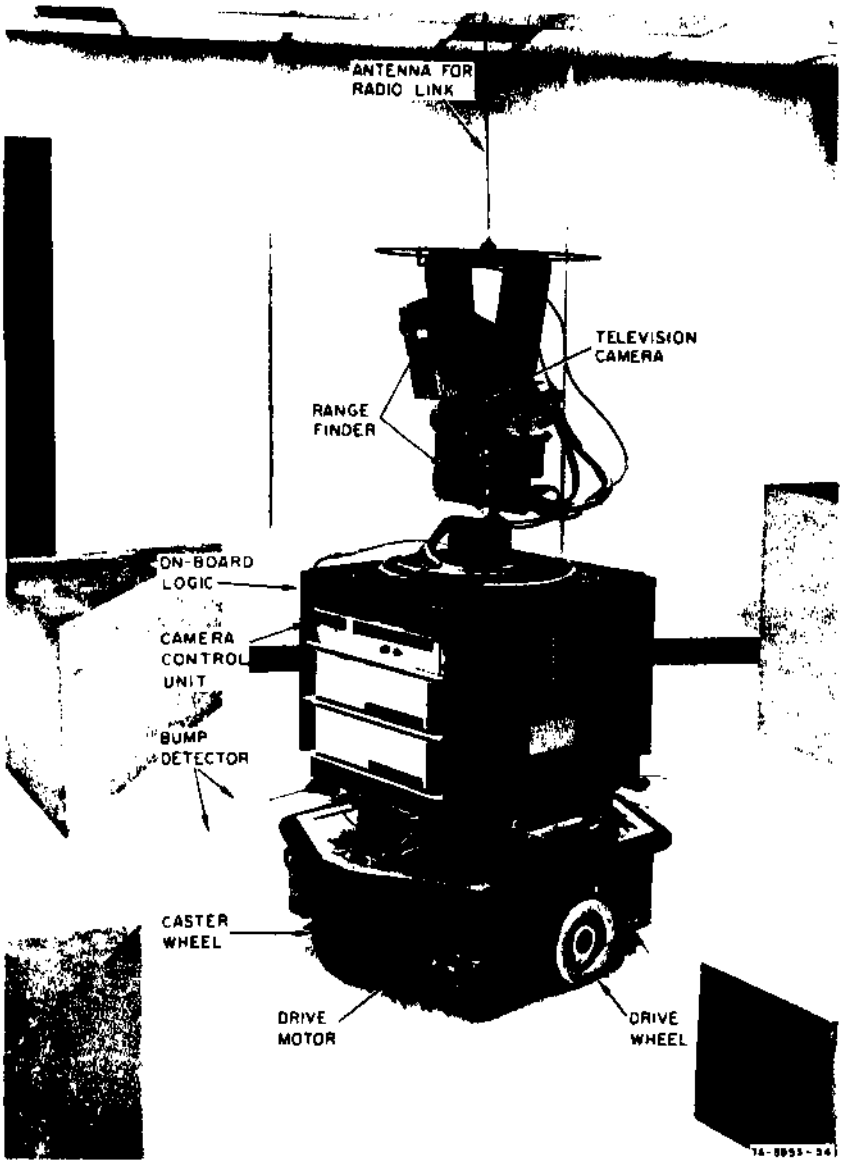

FIG. 1 THE ROBOT VEHICLE

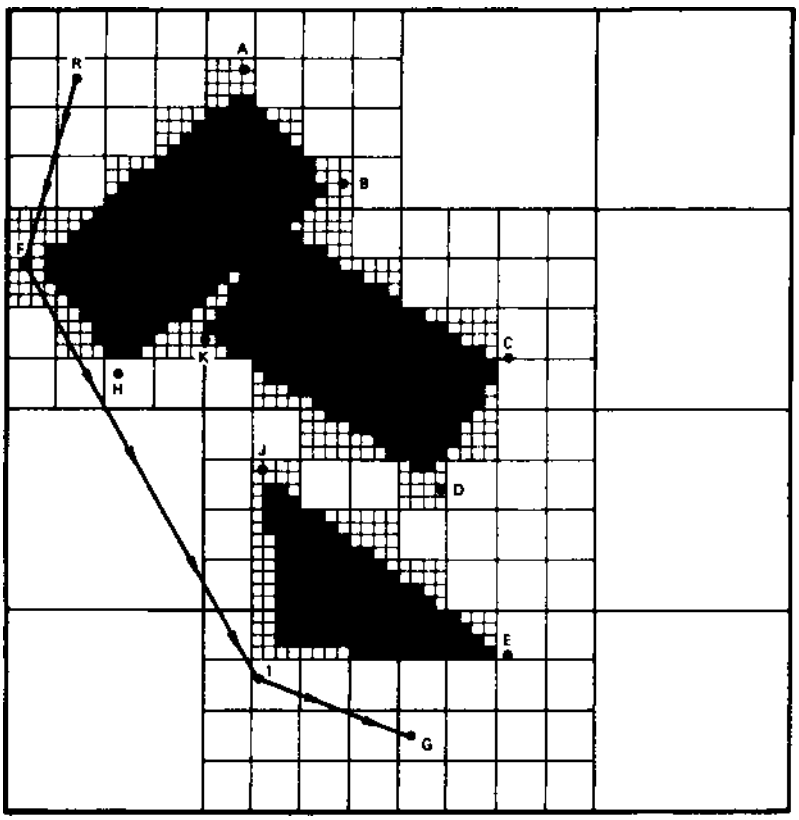

FIG. 2 A GRID MODEL OF A ROOM WITH THREE OBJECTS

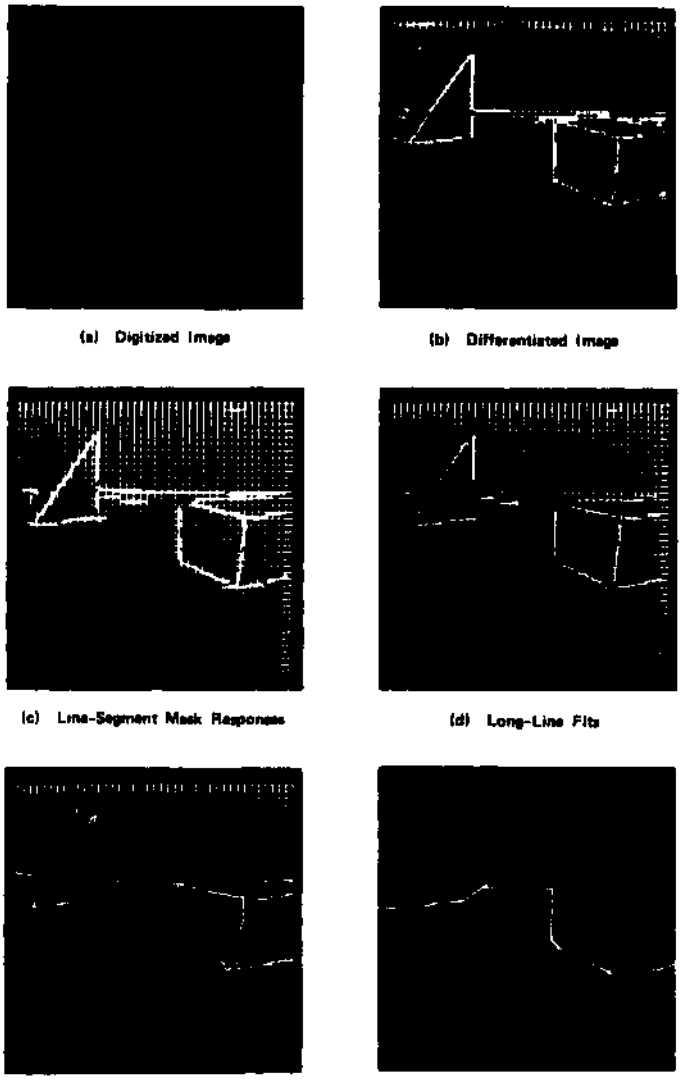

FIG. 3 PICTURE PROCESSING BY THE LINE-DRAWING PROGRAM

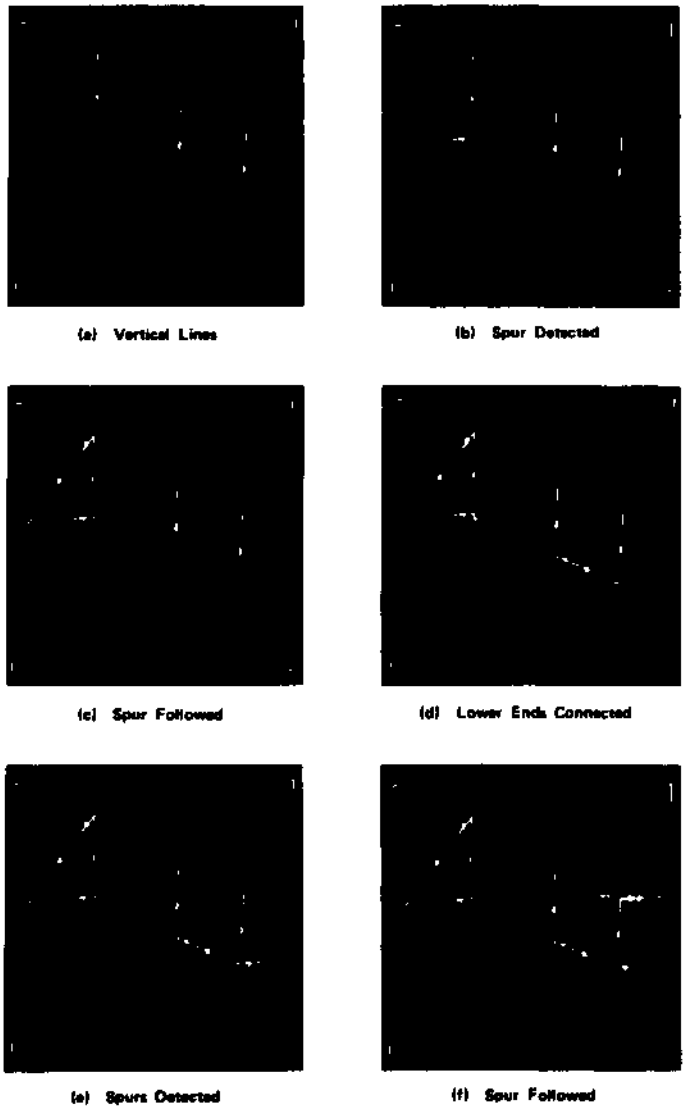

FIG. 4 PICTURE PROCESSING BY THE OBJECT IDENTIFICATION PROGRAM