AD NUMBER: AD0827938

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

This document is subject to special export controls; 1 Jan 1968, and each transmittal to foreign governments or foreign nationals or their representatives may be made only with prior approval of RADC (EMIRC), Griffiss AFB, NY 13440

AUTHORITY

ST-A RADC, USAF LTR, 17 SEP 1971

RADC-TR-67-657

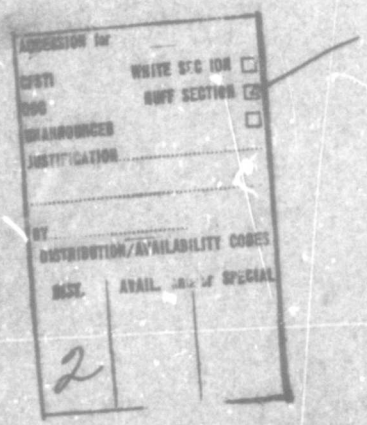
APPLICATION OF INTELLIGENT AUTOMATA TO RECONNAISSANCE

Edited By
C. Rosen
N. Nilsson
Stanford Research Institute

TECHNICAL REPORT NO. RADC-TR-67-657
January 1968

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, New York

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded, by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.



Do not return this copy. Retain or destroy.

BLANK PAGE

APPLICATION OF INTELLIGENT AUTOMATA TO RECONNAISSANCE

Edited By
C. Rosen
N. Nilsson
Stanford Research Institute

AFLC, GAFB, N.Y., 16 Feb 68-142

FOREWORD

This is the Third Interim Report by Stanford Research Institute, Menlo Park, California, covering the period 18 March to 17 December, 1967 on Contract AF 30(602)-4147, Project 4594, Task 459405, for Rome Air Development Center, Griffiss Air Force Base, New York. Captain Stanley Smillie, EMIRC, was the RADC Project Engineer.

Major contributors to the main body of this report were L. Chaitin, B. Jackson, R. Kling, N. Nilsson, B. Raphael, and J. Wensley. Appendices A and B were written by V. Lieskovsky and E. Shapiro, respectively. Professional personnel associated with the various tasks on this project during the past nine months were as follows: Task 1, Visual System, G. Forsen, Leader, R. Bobrow, H. Chan, B. Jackson, and S. King; Task 3, Programming, B. Raphael, Leader, J. Armstrong, J. Bell, L. Chaitin, J. Harris, B. Jackson, P. Lipman, and J. Wensley; Task 5, Special Equipment, S. Wahlstrom, Leader, J. Baer, K. Gardiner, P. Johanson, V. Lieskovsky, E. Shapiro, and J. Yarborough; Task 6, Basic Studies, N. Nilsson, Leader, L. Coles and R. Kling; and Task 7, Program Management, C. Rosen, Leader, M. Adams and N. Nilsson.

Information in this report is embargoed under the U.S. Export Control Act of 1949, administered by the Department of Commerce. This report may be released by departments or agencies of the U.S. Government to departments or agencies of foreign governments with which the United States has defense treaty commitments. Private individuals or firms must comply with Department of Commerce export control regulations.

This technical report has been reviewed and is approved.

Approved: Stanley F. Smillie

STANLEY F. SMILLIE
Project Engineer
Recon Exploitation Section

Approved: for WILLIAM B. MOORE
Chief, Recon Intel Data Handling Branch
Intel & Info Processing Division

ABSTRACT

This report describes the results of research during the past nine months on the project, "Application of Intelligent Automata to Reconnaissance." The primary goal of this project is to investigate techniques in artificial intelligence applied to the control of a mobile automaton in a realistic environment. The main emphasis is on the design of a hierarchy of computer programs that will accept visual and other sensory information from the automaton and direct its actions toward the completion of missions requiring the abilities to plan ahead and to learn from previous experience.

The vehicle and special hardware (except for the visual preprocessor) have all been completed and checked out. The preprocessor has been completed and checkout is now beginning. A large body of software has been written to control the vehicle, and preliminary experiments have been conducted in which the computer has generated commands that the vehicle has successfully executed.

Since the special hardware constructed for the project has been described in detail in previous interim reports, the emphasis in this report is on explaining the software being written to control the automaton system.

CONTENTS

ILLUSTRATIONS

I INTRODUCTION

A. Description of the Project

The primary goal of this project is to investigate techniques in artificial intelligence applied to the control of a mobile automaton in a realistic environment. The main emphasis is on the design of a hierarchy of computer programs that will accept visual and other sensory information from the automaton and direct its actions toward the completion of missions requiring the abilities to plan ahead and to learn from previous experience.

The project began in March 1966 and, since that time, two interim reports describing the first year's work have been published[1,2]* as well as a short paper[3] that presents an overview of the project. In this report, we shall discuss the work performed since March 1967, and refer the reader to the relevant sections of the previous reports for background.

We have constructed an experimental mobile vehicle that is controlled by an SDS-940 computer system over a cable link (soon to be replaced by radio). Figure 1 is a photograph of the vehicle. This vehicle is discussed in detail in the First Interim Report[1] (pp. 56-67) and in the Second Interim Report[2] (pp. 27-31). More information on its mechanical design can be found in Appendix A of the present report. The SDS-940 computer complex is described in the First Interim Report (pp. 11-15). In addition, other special hardware has been planned and constructed, principally a TV preprocessor that is described in the First Interim Report (pp. 68-91 and pp. J-1 through J-7) as well as in a paper by Forsen.[4]

Most of the research content of this project concerns the development of concepts and software necessary to control the automaton. On this subject, the present report presents a more or less self-contained review of our present status.

* References are listed at the end of this report.

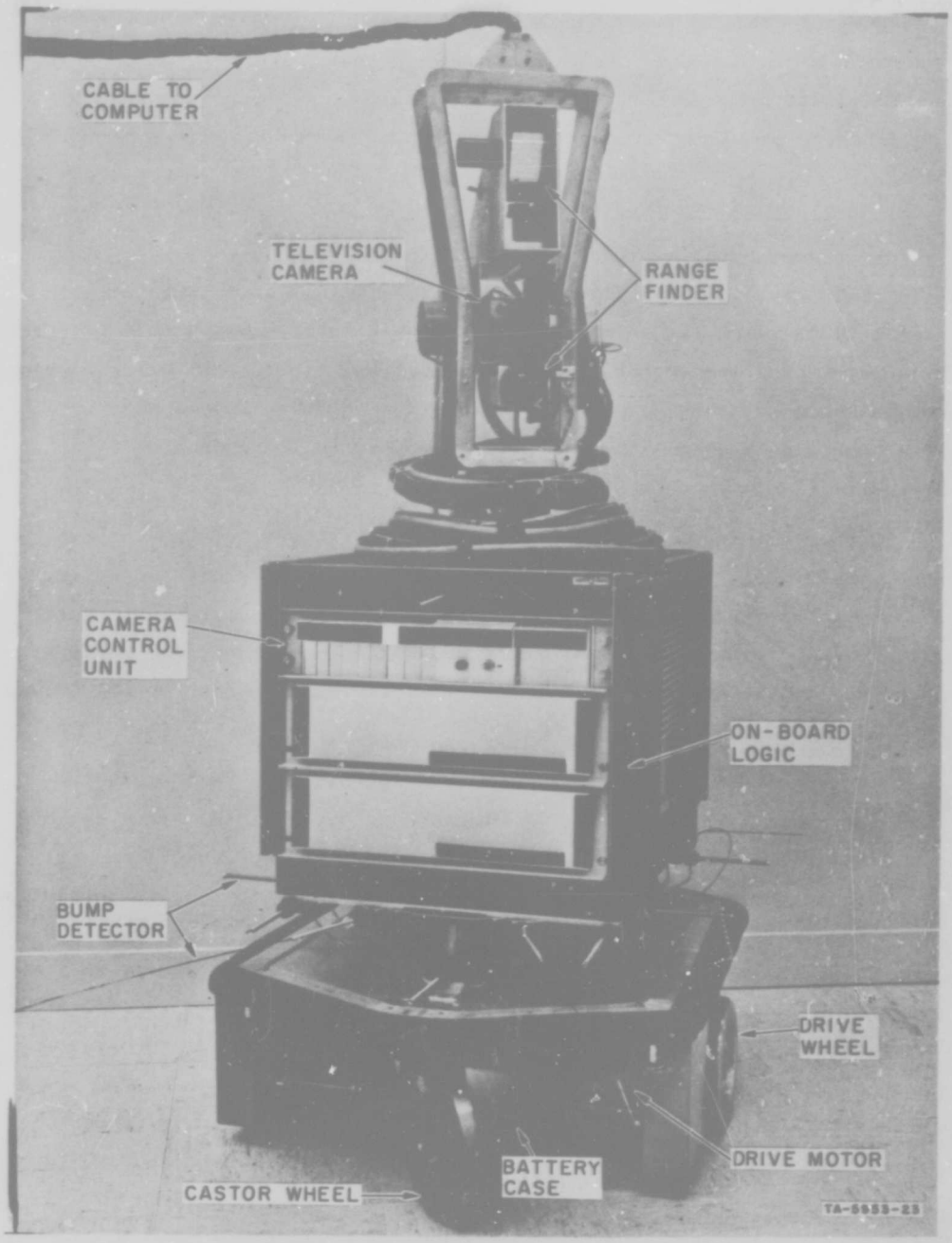


FIGURE 1. AUTOMATON VEHICLE

Recently SRI proposed to augment and extend the present project to undertake more thorough research in such component areas of the project as (1) visual processing, (2) high-level specification languages, and (3) enhanced problem-solving abilities. This augmentation has recently begun.

B. Status of the Project

Since the last interim report[2] in March 1967, progress on the construction and checkout of special equipment for the automaton project has been somewhat slower than expected, while progress on the software has been on schedule. All contemplated special equipment except the TV preprocessor and the radio link has been completed, and an extensive body of software has been written to allow experimenting with the automaton vehicle. The project, including the debugging of current software, has recently been slowed by hardware problems with the SDS-940 computer system. It is expected that these problems will soon be successfully solved.

In December 1967, we expect to augment the core memory of the computer by an additional 32 K. This extra core will permit time-shared use of the SDS display, which is an important tool in improving the programs for the automaton visual system. Sometime in April 1968, we expect to take delivery of a Bryant 96-million character disc system on which to store user files. The present tape-oriented system causes lengthy file-retrieval delays when several people are using the computer.

The automaton vehicle (with its action logic modules and associated equipment), the Fixed Control Unit and interface with the computer were completely checked out and operational by 1 September 1967. Preliminary tests of all this equipment under actual computer control were made prior to the difficulties with the computer. These tests confirmed that the vehicle can be positioned and moved accurately by the computer.

A description of the major elements in the communication chain between the vehicle and the computer is contained in the Second Interim Report[2] (pp. 29-31). The access to this communication chain is through

BLANK PAGE

the Fixed Control Unit (FCU). The console of the FCU permits the experimenter to directly control the vehicle without using the computer (when this is desirable). This console and its operation are discussed in detail in Appendix B, which is attached to this report.

Although the construction of the TV preprocessor was completed last May, checkout of this equipment was given lower priority than that of the other special equipment and, as a result, has only recently begun. A decision has been made to postpone indefinitely the construction of arms for the vehicle,* so the only remaining equipment to be added is a radio link to replace the present cable connection. Work on this radio link has begun.

The automaton software system seems to be progressing very well. We have successfully made changes to the SDS-940 executive system to permit real-time control of the automaton under time-sharing. In addition, a special executive program has been completed. This program, termed the VALET, allows one FORTRAN program to dismiss itself and start up another. The VALET handles communication of data among the FORTRAN programs via the Rapid Access Data (RAD) file. Future versions of the VALET will permit similar communication between LISP and FORTRAN.

In addition to these systems programs, a large amount of software has been written that enables an experimenter to give the automaton a command such as "GOTO X Y," where X and Y are coordinates of a desired goal position. These programs will be summarized briefly in Sec. II and then discussed in detail in the remainder of this report.

---

* A possible design for arms was discussed in Appendix B of the Second Interim Report (pp. 53-57).

II AUTOMATON SOFTWARE ORGANIZATION

In the previous interim reports, it was suggested that the programs controlling the automaton be organized in a hierarchy of four levels. The lowest, or immediate-action level, would actually activate the motors and sensors. A second, or tactical level, would be responsible for solving the simple navigational problems of locomotion. A third, or strategic level, would be able to direct the automaton in more complex tasks involving finding and pushing objects around in its environment. The highest, or problem-analysis level, would translate an input command (stated in a suitable specification language) into a sequence of tasks to be performed by the strategic level.

Since the last interim report, the actual programs for the immediate-action level have been completed, and enough has now been accomplished on the tactical level to allow us to describe in detail how the first version of it will function.

The neat division into hierarchical levels which remains a convenient way of discussing the behavior of the overall software system, was not always rigidly adhered to in the writing and organization of the actual programs.

In this section, we shall discuss the major existing blocks of automaton software and data structures and how they relate to each other. These, in turn, will be discussed in greater detail in subsequent sections. A block diagram of the overall system is shown in Fig. 2.

At the highest level are a set of executive routines written in FORTRAN. It is through these routines that an experimenter will use the automaton system. One routine initializes the system by setting the automaton's map of the world (to be described below) to a preset form. Alternatively, this routine might instruct the automaton to take various television views of its immediate surroundings, store the information

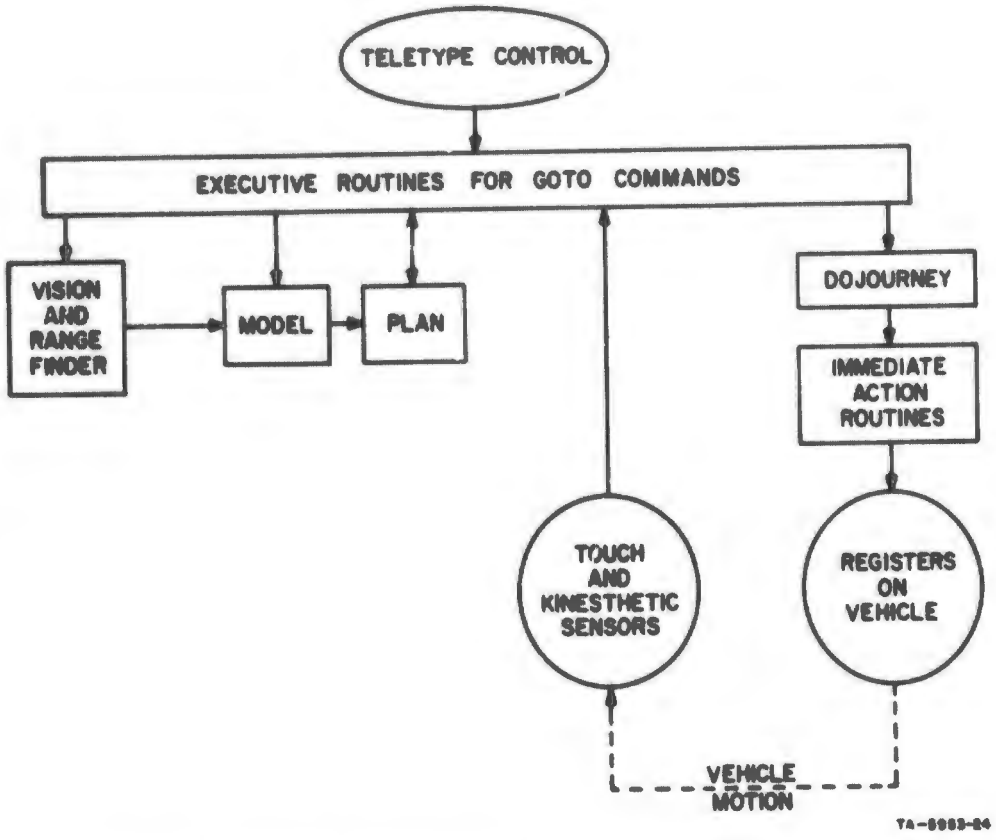


FIG. 2 CURRENT AUTOMATON-SOFTWARE ORGANIZATION

thus obtained in its map of the world, and align itself in some appropriate initial position with respect to its surroundings.

After initialization, the automaton is ready to perform some task such as GOTO POSITION (X_1, Y_1). A program called the GOTO program directs the performance of such a task. Several GOTO programs exist, and probably these will continue to be modified and to proliferate. An executive routine selects and monitors the execution of these GOTO programs. We shall discuss typical ones in more detail in a following section.

The automaton's respository of knowledge about its surroundings is its map of the world or model. Actually the model consists of several data items giving such information as which parts of the environment are filled or unfilled and the location and orientation of the automaton and its articulated parts. Future versions of the model will also contain the names and properties of various objects in the automaton's environment. The information in the model can be suplemented or altered directly by the experimenter through one of the FORTRAN executive routines, or, more naturally, by the automaton's own experience as sensed by the vision and touch systems.

Much of the range finder and visual systems have been described in detail in other reports. (See especially a paper by Forsen.[4]) We shall summarize some of this material below but will concentrate on the higher-level programs within the visual system. In the organization of the software blocks, we had the option of allowing the executive system to receive sensory reports directly from the visual system or to receive visual sensory information only via the model. We selected the latter alternative for the first version of the software since we will be wanting to store all of the information extracted by the visual system in the model anyway.

The touch sensor operates through an interrupt system to inform higher programs when the automaton has bumped into something. In addition special programs keep track of the pan and tilt angles of the television camera, the status of the iris and focus settings, and of the automaton's own location in its environment. The automaton's location is "dead-reckoned" by keeping track of wheel rotations. This location information,

which is subject to error due to slippage, must be updated occasionally by information from the touch and visual systems. The development of effective reorientation techniques is an important area for future study.

A major routine used by the GOTO program is a FORTRAN program called PLAN. PLAN accepts as input the automaton's present location and the goal location and delivers as output a sequence of subgoal locations ending with the goal location. Each subgoal is on a straight-line path from the previous subgoal. PLAN uses the model to compile such a journey. After receiving a journey from PLAN, the executive GOTO program may call the routine DOJOURNEY. DOJOURNEY is a FORTRAN program that accepts as input the sequence of subgoals in a journey and calls the appropriate assembly language immediate-action routines to drive the motors.

The programs that are summarized in this section and discussed in detail in the following sections are either completely written or specified to the point where only routine coding remains. Higher-level programs that could be considered to be at the strategic or problem-analysis level have not yet been specified. Extensive work on these programs shall not be attempted until some experience is gained with the lower-level programs.

III THE MODEL

The two most important data structures in the model are the grid model and the line model. The grid model is a hierarchical arrangement of cells that subdivide the automaton's environment to an arbitrary degree of precision. The units of the grid model, called "cells," each contain information about a square area of the environment; each cell is divided into a 4 × 4 array of "squares."

The top-level or main cell of the grid model represents the entire environment. Each of its squares may be marked "unknown," "full," "empty," or "partly full," reflecting the automaton's knowledge of the particular area represented by the square. Alternatively, the square may be divided further, i.e., its area may be represented by another complete grid cell that is in turn divided into a 4 × 4 array of squares.

Clearly, the modeling system need not divide the square for a given area if it is known that that area is completely full or completely empty; the division of such a square could not provide any further information. If, however, a square is partly full (and if it represents any significant amount of the environment), it should be subdivided; the grid cell so produced will then contain information about just which parts of the square area are full. Thus, this selective subdivision preserves the same knowledge of the environment that would be included if all of the model was uniformly, finely divided. In addition, selective subdivision has the advantage of being far more economical of storage than the uniform subdivision would be.

It is, however, more difficult to plan journeys by using the grid model than by using a fully divided large array. Consequently, we use 25, not 16, words of storage per cell and keep in the extra 9 words additional information to help programs using the grid model. The words are filled as follows:

(1) Depth of the cell in the model. The top-level cell is at depth 1, its immediate descendants are at depth 2, etc. Executive routines, which initialize the automaton system, specify a maximum depth for the model, below which cells will not be created. (This parameter tells the system how small an area of the environment is still considered significant.) At the maximum depth (and only at that depth) partly full squares are not divided but are marked partly full.

(2) X-coordinate of the lower left-hand corner of the cell

(3) Y-coordinate

(4) Z-coordinate (at present we use only 0 for Z values)

(5) Size is the length of a side, in units internal to the grid model of one of the 16 squares within the cell. (The modeling units are specified by the size value for the top grid cell; this value is a parmeter provided by automaton-initializing programs. The scale factor required to convert model units into units of the real world is another system parameter.)

(6) A pointer to the parent grid cell

(7) A pointer to the parent square (that one of the 16 squares within the parent cell to which the new cell corresponds).

(8) Available for local use within subprograms

(9)

(10) to (25) Data about the squares within the cell. An individual datum either is a marker for unknown, full, empty, or partly full or else is a pointer to another cell.

When the automaton bumps into an object at some point (X,Y), a fairly simple mechanism records that bump in the grid model. Dividing squares as necessary, a program traces down through the model to find the lowest (allowable) level grid square that contains the point (X,Y); the program then marks that square as partly full.

A more complicated mechanism incorporates into the model information obtained through the visual system. After preprocessing and least-squares fitting (to be described in the next section), the automaton's image of a scene might look like that shown in Fig. 3. Then, another program will clean up the picture, by connecting segments which seem to be colinear, by forming corners, and by removing short, unconnected segments (see Fig. 4).

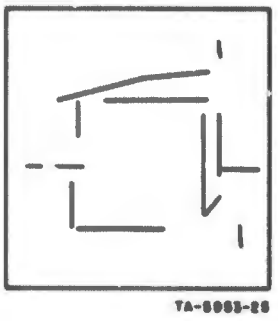


FIG. 3 THE RESULTS OF INITIAL PICTURE PROCESSING

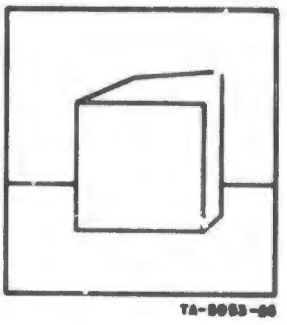


FIG. 4 THE PICTURE AFTER FURTHER PROCESSING

Next, using information about the location and orientation of the automaton and about the particular camera lens being used, a program will translate the locations of lines and objects in the picture into locations in the automaton's environment. Finally, a routine will determine what the picture implies about full and empty areas of the floor. For example, the striped area in Fig. 5 is definitely empty and the solid area is definitely full (assuming convex objects).

Nothing more is assumed, The object could be a cube, in which case more floor area might be marked as full, but it could also be a wedge with a triangular base as shown. Picture processing is not yet so exact

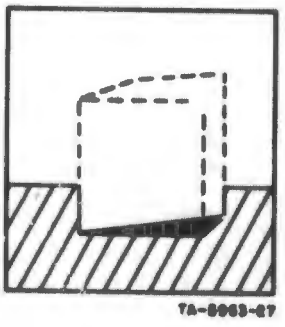


FIG. 5 FULL AND EMPTY REGIONS

as to guarantee the existence of the apparent corner at the back left of the top surface of the object; our policy now is to avoid incorporating very uncertain information into the model.

The information obtained from such picture analysis is incorporated into the grid model by another set of routines. These programs accept as input a list of coordinate pairs; the pairs define a polygonal area. The routines also accept a status word (full or empty). They trace through the grid model, determining which squares are at least partly contained in the polygonal area. Squares completely contained in the area are marked with the status word; those partly contained are subdivided as necessary, and then their descendants are considered.

The grid model is particularly suited for use in journey planning, when the automaton is really just concerned with full and empty areas. A program tells the journey-planning routine which cells a proposed journey leg intersects, and the planning routine keeps or discards the proposed leg accordingly as those cells are empty or not. The grid model is not, however, at all well suited for certain other calculations. For example, a full square area of the floor (Fig. 6) might appear in the grid model as in Fig. 7 as a result of the quantizing process.

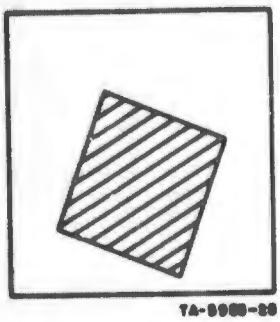


FIG. 6 A SQUARE FLOOR AREA

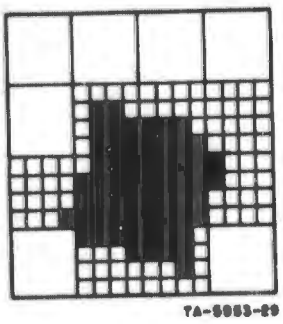


FIG. 7 THE GRID-MODEL VERSION OF THE SQUARE AREA

Clearly, right angles (or any corners) are difficult to find in the grid model; the automaton would have trouble if it tried to find a cube by first looking in the grid model for a full square area of floor space.

The system will, however, have much more exact information, obtained from the visual pictures, about corners and lines in the floor plan. We have decided to keep this information in a (simpler) model, the line model. This model will just be a record of the locations (in grid-model units) of all lines found in pictures; maintenance programs for this model will be used to combine information from several pictures. For example, various pictures might have provided the lines in Fig. 8 (and the shaded areas of the figure would have been marked full in the grid model). These lines would be cleaned up to give the lines in Fig. 9.

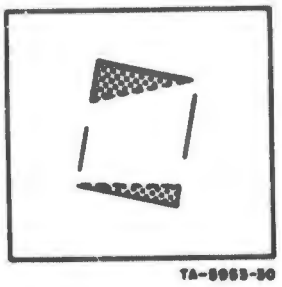


FIG. 8 A PORTION OF THE LINE MODEL

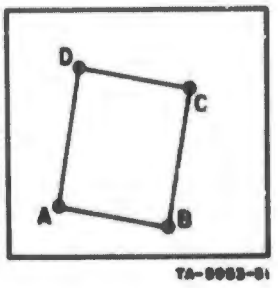


FIG. 9 THE LINE-MODEL PORTION AFTER MAINTENANCE ROUTINES HAVE BEEN CALLED

We will also use the line model to update the grid model. After performing the maintenance shown in Figs. 8 and 9, the system would find a closed loop of lines; it could therefore tell the model-filling routines to mark the area ABCD full (or empty, depending upon the previous marking) in the grid model and to discard unnecessary subdivisions in that model.

The line model will also contain a list of vertical lines, so that the automaton should be able to handle requests like "find a cube at least 3 feet high."

In addition to the grid and line models that are being implemented, we plan to start work soon on a property model. This model, which will be maintained by programs in the LISP language, will contain symbolic list structured descriptions of the environment. For each object type, there will be at least a list of characteristics necessary and sufficient for classification of some particular object as belonging to the given class. A cube might be marked as having four corners, four right-angles-in-base; further, for each property, there would be an indication of what FORTRAN routine to call to check for that property. A second part of this data base might include further useful information about the objects. For example, it might tell where in a room to look for the objects of a given type, and it might tell which of several pushing procedures should be used for the object.

LISP programs will maintain a list of fully known objects, including at least such information as type, size, and location. It will also maintain a list of partially-known objects, objects not yet identified (either because there has been no reason for going through the identification procedure or because there isn't enough known information for positive identification). For each object on this list, those known properties will be listed.

There will certainly have to be a mechanism for changing the status of an object from unknown to known. Probably in the early programs this should be the only type of status change allowed--i.e., identification, once made, should be considered final (and the necessary-for-identification property lists in the data base should be sufficiently complete to make this procedure reasonable).

IV THE VISUAL SYSTEM

Visual processing for the automaton consists of the following five steps performed in sequence:

(1) Digital sampling of video signal from the TV camera on the vehicle

(2) "Differentiation" of digitized picture to accentuate intensity gradients

(3) Extraction of short line segments at various orientations from the differentiated picture

(4) Connecting the oriented line segments to form connected groups

(5) Least-squares fitting each connected group by one or more lines

Special purpose hardware has been designed and constructed to perform the first three steps extremely rapidly (in about 1/10 second). The other steps are implemented by special programs. The hardware has not yet been checked out, however, so in the meantime its functions are being simulated by software. Details of the operations in the first three steps have been thoroughly described in previous reports. In this section we shall quickly summarize these operations and discuss at more length Steps 4 and 5.

The current visual system first produces a 120 element by 120 element array of intensity readings. These original intensity readings are then converted to differentiated readings by use of the formula[8] $|I(x,y) - I(x+1, y+1)| + |I(x+1, y) - I(x, y+1)|$, where $I(a,b)$ means the intensity reading at point (a,b), $1 \leq x,y \leq 120$. This differentiation formula gives the sum of the intensity differences of the endpoints of each of the lines in Fig. 10. This sum is a scalar quantity and does not indicate direction. If the value exceeds a threshold, the direction of the gradient at that point is determined by subsequent processing.

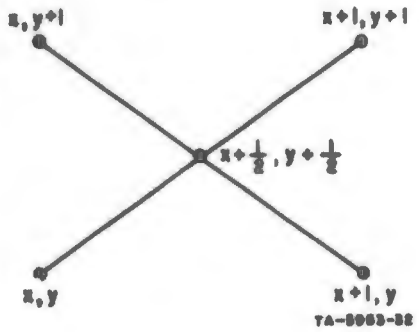


FIG. 10 DIFFERENTIATION OF THE DATA

In processing a differentiated picture, the visual system first calculates the correlations between relevant 7 X 7 subfields and the masks shown in Fig. 11. For each of the numbered lines in Fig. 11, the quantized mask, corresponding to the line, consists of those squares through which the line passes.

For each 7 X 7 subfield, the total response along each of the sixteen mask lines is calculated. Thus, the response along mask 8 is the sum of the responses at points (1,3), (2,3), (3,4), (4,4), (5,4), (6,5) and (7,5) (Fig. 12). The first twelve masks are evenly spaced about the center (adjacent pairs of them are 15^0 apart); the last four masks were added to provide more resolution between near-horizontal and near-vertical lines.

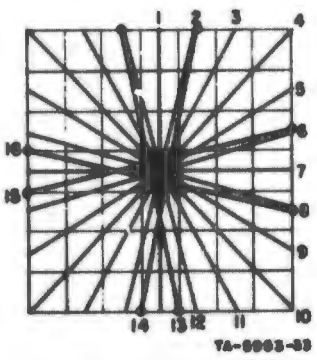


FIG. 11 MASK RESPONSES

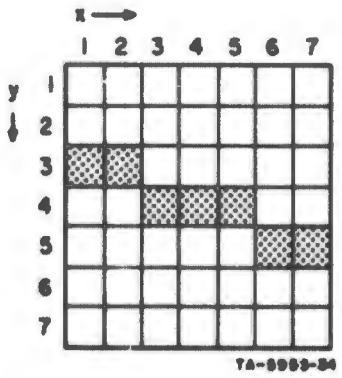


FIG. 12 CALCULATION OF THE RESPONSE ALONG MASK 8

The maximum of the sixteen sums along the masks is compared with the average response for the sixteen masks. If the difference between the two values is not greater than the greater of a preset threshold and one determined by the average local contrast, the correlation for the subfield is zero. Otherwise, the nonzero correlation, the location, and the number of the mask that gave the maximum response are saved.

During this part of the process, the system shrinks the picture to 38 X 38 elements. It does so by dividing the interior of the picture (all points that are the centers of 7 X 7 subfields) into 3 X 3 squares (see Fig. 13). Each such square is then given the maximum of the correlations of its nine points. The system also remembers the position within the 3 X 3 square of the point giving this maximum (if there is, in fact, a nonzero correlation).

Next, the system connects adjacent squares (of the 38 X 38 picture) that have nonzero correlation values and whose mask responses are "consistent." For each point, only the four adjacent squares shown in Fig. 14 are considered. (This procedure avoids duplications, provided, of course, that none of these adjacent squares is off the end of the picture.)

In determining consistency, the system considers the masks giving maximum correlations and also considers the locations of the points whose 7 X 7 subfields gave these maximum responses. If the maximum mask responses for an adjacent pair of squares are both sufficiently close to

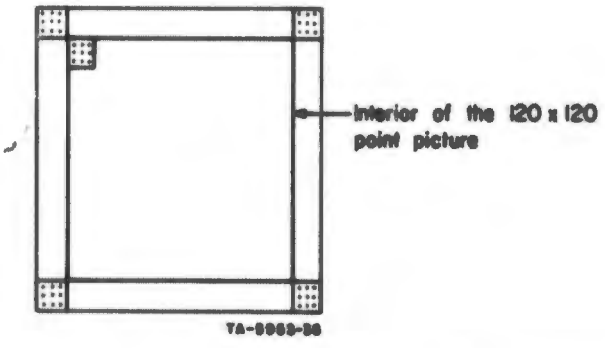


FIG. 13 THE INTERIOR OF THE PICTURE DIVIDED INTO 3 × 3 POINT SQUARES

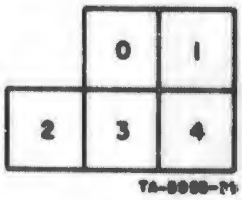


FIG. 14 A GIVEN POINT, 0, AND ITS FOUR ADJACENT POINTS

(within a preset tolerance of) the line between the two responding points, the adjacent pair is included in a list of connected squares. For example, the first pair of squares in Fig. 15 would not be considered connected, although the second pair would.

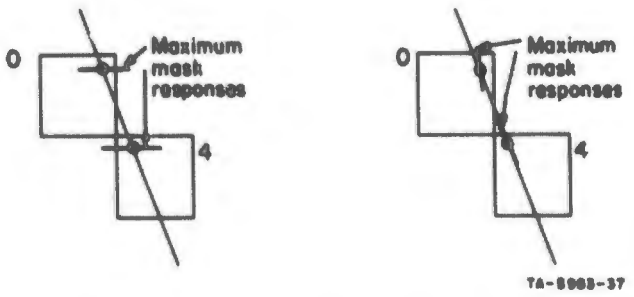


FIG. 15 UNCONNECTED AND CONNECTED PAIRS

In order to increase the efficiency of this portion of the program, the system uses a previously-calculated (9 x 9 x 4) table telling it, for each pair of points (from the nine points per square) and for each of the adjacent squares 1-4, which mask responses are consistent.

Finally, the list of connected points is divided into connected groups; each pair of points in such a group can be joined by some list of connectioned adjacent pairs. Least-squares fitting is applied to such groups.

In the least-squares portion of the program, first the points in a group are ordered by their projections along some line selected as a first approximation. The program then discards that line and incorporates the ordered points, in some prescribed incorporation order, into a least-squares fitted line. The program includes in the line only those points that are within a "reasonable distance" of the growing line. The reasonable distance changes as the line grows; after the first few points have been incorporated, this tolerance level is the maximum of the average deviation so far and of a preset minimum tolerance. After extracting one line from a connected group, the program will try to fit another line to the remaining points, if there are enough leftover points.

We are currently experimenting with ways of ordering the points and of ordering their incorporation into the growing line. One attempted scheme ordered the points along the line between the extreme points in the group; it then incorporated the points in this same order. We feel,

however, that the extreme points of a group are particularly apt to be stray points. Thus, we did not want such points to determine a preliminary ordering or, even more importantly, to be the first points incorporated into a line, determining the line's initial orientation.

Therefore, we are now considering a scheme that uses the most common maximum mask response in a connected group to give the direction of the line along which the points are ordered. The points are then incorporated in this same order, with the exception that the first few points are included last, not first. Before definitely accepting this scheme, we want to examine its effects on a large body of data. We hope that it will correctly find lines such as that in Fig. 16 and will discriminate between the two lines in Fig. 17.

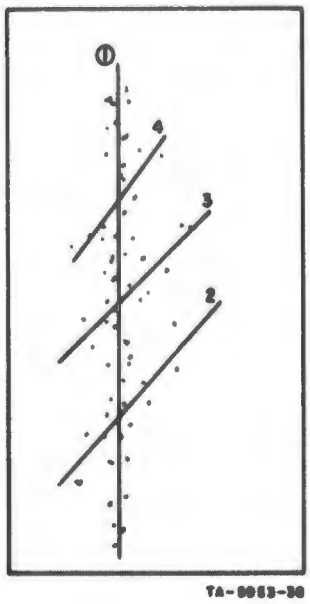


FIG. 16 A GOOD LEAST-SQUARES SCHEME SHOULD FIND LINE 1, NOT LINES 2 - 4

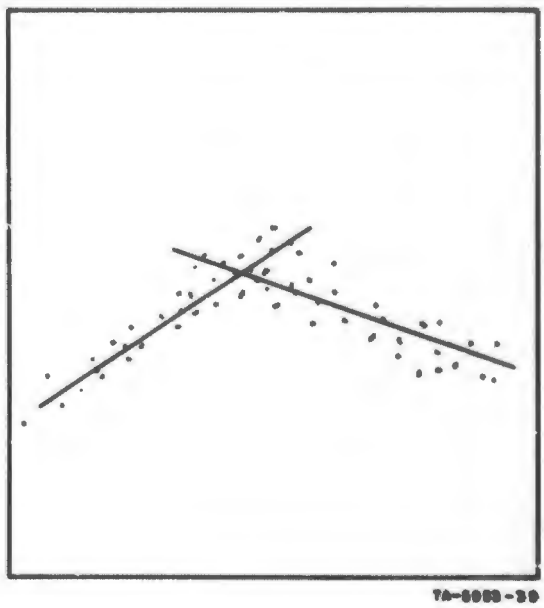


FIG. 17 A GOOD SCHEME SHOULD ALSO RECOGNIZE THE TWO LINES IN THIS SCENE

V JOURNEY-PLANNING AND EXECUTION ROUTINES

This section describes the FORTRAN programs that have been written to plan and execute locomotion. These programs which are called PLAN and DOJOURNEY, respectively, are both callable from executive routines such as GOTO.

A. PLAN

The problem for which the PLAN routine is used is the preplanning of a journey from the automaton's present position (R) to a goal position (G) through an environment known to contain empty space, objects, and also regions whose contents are unknown. PLAN uses the information about the environment that is stored in the grid model described in Sec. III above.

The basic process involves generating and searching a decision tree that represents at each node the possible alternative next moves to make from the position which corresponds to that node. This process requires that a method exist for defining "successors" of a node (i.e., a set of nodes representing positions that are directly accessible from the position corresponding to the node) and also a method for searching the tree in a way that is in some sense efficient. These two parts of the problem will be treated separately.

1. Successor Generation

Each node of the tree corresponds to a subgoal in a possible journey. The successors of a node are determined by reference to the information stored in the grid model. If the model reveals that the goal is directly accessible from a subgoal, then the goal is that subgoal's only successor. Otherwise the model is examined to find the first obstruction that would be encountered on a straight-line journey to the goal. Two positions are then found, one to the left and one to the right of this obstruction.

These positions are "candidate" successors and are held in a list. At the same time the obstruction is marked as having been already considered.

The list of candidate successors is now examined, and the model is searched to find a straight-line path to each of those candidate positions. If successful, the candidate is made a successor and is removed from the list. If the candidate cannot be reached and the closest obstruction on the path to that candidate has already been considered, the candidate is rejected and is removed from the list. If the closest obstruction has not yet been considered, the candidate is removed from the list and the positions to left and right of the obstruction are added to the list.

The process is continued until the list is exhausted. All positions that are generated as successors of a node can be reached by straight-line travel from that node.

A means is provided whereby "unknown" regions in the grid model may be treated as full or as empty.

2. Searching the Tree

The basic method that will be used for generating and searching the tree is outlined in the following section.

We start by generating the successors of the present position, R. From then on, for each node that has not yet had its successors generated, we calculate a number that is used to decide which node should have its successors generated next. This number is the sum of the length of the path (D) from R to the position corresponding to the node plus an estimate (E) of the length of the remainder of the journey. In considering which node to investigate further (by finding its successors), that node is chosen that has a minimum D + E. When the node with minimum D + E is the goal, the process terminates. This method of searching the tree will find the journey of shortest length if E is a lower-bound estimate of the actual remaining distance from a node, and will do so by searching a smaller part of the tree than other methods, under a set of assumptions that are reasonable for most journeys.[6] For the estimate (E) we are

using the straight-line distance between the node and the goal, which is a lower bound on the actual path length.

The above process is halted if the tree search does not terminate within a specified time (or number of steps). If it is halted, the best journey to the goal so far found (if any) can be used.

If the model contains significant parts that are unknown, then there are times when it can be disadvantageous to attempt journeys across such regions. At other times it may be desirable to go into unknown areas, for example when time is available to do some exploring during a journey. A parameter can be set that is a weighting for all parts of a journey through unknown space. If this is set to 1, unknown space will be treated as empty. For values above 1, the lengths of paths going through unknown space is proportionally increased, and such regions tend thereby to be avoided (though not entirely prohibited, unless the weighting is set to infinity). If the weighting is less than 1, then unknown space is preferred to empty space, and "exploring" is automatically preferred.

3. Alternative Tree-Searching Methods

Finding the most efficient strategy for planning and executing a journey is a much more complex matter than finding optimal tree-searching techniques. The method described above is guaranteed, under certain reasonable assumptions, to find the shortest route, and in so doing will consider the minimum number of nodes in the tree. There are, however, cases where this procedure is not maximally efficient. We will consider some of these.

If, as in our case, the computation time and the automaton moving time can be overlapped, a strategy can be more efficient if it starts the automaton on its way before the final journey has been completely planned. This is particularly true of movement through a region containing a large number of small obstructions. As an illustration, a strategy that considered all obstacles in New York before leaving San Francisco on a cross-country drive would delay the journey and thus

possibly impair efficiency. Tree searching in that case should go some depth into the tree, then start the actual movement on the path guessed to be most promising.

In practice, the best strategy may be one in which a solution was found that has close to the minimal cost but for which the computing required to achieve this solution is significantly reduced. For example, such a situation arises when a large number of small obstructions exist with large spaces between them. In such a situation, the optimal strategy would consider only one or two obstructions at a time and would not search the tree to maximum depth for the many, almost identical trial solutions.

If a large obstruction is close to the goal, and there is no similar obstruction to the start, it will frequently be the case that the reverse problem (i.e., goal to start) can be considered and a quicker solution found. For example in Fig. 18, planning a journey from R to G, using the successor operator described above, will involve examining many branches around the small obstructions, but this is not so in planning the reverse journey from G to R.

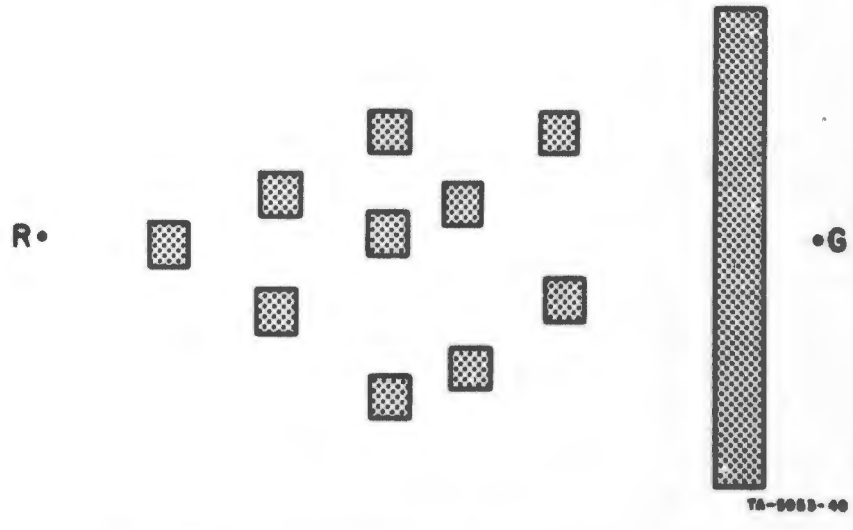


FIG. 18 A CASE WHERE THE REVERSE OF A PROBLEM OFFERS FASTER SOLUTION

B. DOJOURNEY

The program DOJOURNEY accepts as input the list of subgoals on the path generated by PLAN. It begins by taking the automaton vehicle from its present position to the first subgoal. After this has been accomplished, the first subgoal is removed from the list, and the process starts over, taking the automaton vehicle to the next subgoal (if any) on the list. DOJOURNEY uses routines that are able to move the automaton from a given position and heading to an arbitrary position reachable from the given position by a straight-line (obstacle-free) path. These routines first activate wheel motions to begin the vehicle moving in the right direction and then activate both wheels simultaneously to move the vehicle the required distance. Automaton wheel motions are actually activated by calls to immediate-action level programs, described in the following section. If, during execution of DOJOURNEY, the automaton vehicle bumps into some obstacle, an interrupt is sent up through the immediate-action program. DOJOURNEY advises the executive routine of this event so it can take appropriate steps.

VI IMMEDIATE-ACTION ROUTINES

The immediate-action routines are a set of programs, written in assembly language, that can be called by higher-level FORTRAN programs such as DOJOURNEY. From the point of view of the higher programs, each of the immediate-action routines is a command that is automatically executed when called. These commands control the action logic modules (ALMs) on the vehicle which in turn control the motors and are used to read sensory information.

The commands are divided into two classes. Class A commands are those that generally cause something to happen on board the vehicle or read sensory information. Class B commands do "housekeeping" type operations.

A. Class A Commands

Calls to class A commands are of the form: CALL CMND (N,HANDLE). N is an integer to be set prior to the CALL. For action commands, it specifies a magnitude and sign of the desired action. HANDLE is set to zero upon the CALL and will remain so until the action is successfully executed or until an INTERRUPT occurs. When the action is successfully executed, HANDLE will be set to some negative number. Interrupts occur when a motor limit has been hit (for focus, iris, pan, and tilt motors) or when bump switches are closed. When an interrupt occurs, HANDLE will be set to the (positive) number of increments not yet actually completed at the time of interrupt. Additionally, if a bump occurs, N will contain an indicator defining which of the bump switches were closed.

Any action may be stopped by reissuing the command with an N of zero.

The following list defines the major class A commands that have thus far been implemented:

CALL LEFT (N, HANDLE) -16384 ≤ N ≤ 16384

This command moves the left wheel forward (+) or backward (-) N 32nds of an inch. This can get a BUMP interrupt.

CALL RIGHT (N, HANDLE) -16384 ≤ N ≤ 16384

This command moves the right wheel forward (+) or backward (-) N 32nds of an inch. This can get a BUMP interrupt.

CALL MOVE (N, HANDLE) -16384 ≤ N ≤ 16384

This command moves the vehicle (by moving both wheels simultaneously) forward (+) or backward (-) N 32nds of an inch. This can get a BUMP interrupt. The wheels are moved in a "coupled" mode to insure straight-line motion.

CALL TURN (N, HANDLE) -16384 ≤ N ≤ 16384

This command turns the vehicle about its vertical axis (by moving both wheels simultaneously and in different directions) clockwise (+) or counterclockwise (-) N 32nds of an inch. It is left to the calling program to determine the relationship between N and degrees or radians. This can get a BUMP interrupt.

CALL PAN (N, HANDLE) -201 ≤ N ≤ 201

This command moves the camera and range finder N 1.8 degree increments clockwise (+) or counterclockwise (-). This can get a BUMP interrupt. It can also cause a LIMIT interrupt, which says "I can't turn any further in this direction." This LIMIT position is, in fact, exactly to the rear of the vehicle, and occurs from the clockwise or counterclockwise command.

CALL TILT (N, HANDLE) -58 ≤ N ≤ 58

This command moves the camera and range finder, N 1.8 degree increments up (+) or down (-). This could possibly get a BUMP interrupt. It can also cause a LIMIT. Assuming that true horizontal is "zero", the

upper limit is 57.6 degrees (=32) and the lower limit is 45 degrees (N = -25).

CALL RANGE (L, HANDLE)

This command reads the current value of the range finder. HANDLE is set negative upon completion of the read operation and L is set equal to the range in arbitrary units.

CALL IRIS (N, HANDLE) $-142 \leq N \leq 142$

This command moves the iris on the TV camera N increments toward the closed position (-) or toward the full open position (+). This command can cause a LIMIT interrupt.

CALL FOCUS (N, HANDLE) $-522 \leq N \leq 522$

This command sets the lens focus on the TV camera. The focus is moved toward infinity (-) and toward the closest focus limit of 1 foot six inches (+). This command can cause a LIMIT interrupt.

B. Class B Commands

Interrupts are generated by the external equipment when a limit or bump occurs as a result of issuing a class A command. The higher-level FORTRAN program may provide for specific action or cause the interrupt to be ignored. Either choice is handled through the use of the interrupt class B commands. The commands arm the interrupt and set up a return to location L such that if the interrupt signal from the external equipment does in fact occur (generally unexpected), program control will immediately be transferred to location L. Location L will generally be the start location of a routine that will take remedial action with regard to the class A command responsible for the interrupt. It should be noted that the FORTRAN program will be interrupted, if the interrupt has been armed, when the interrupt is received, and the program will not know what instruction was being executed at the time of interrupt. Class B commands may be issued as many times as desired, and control is always transferred to the latest executed case. Recall that if an uncompleted action does

occur, HANDLE associated with the interrupt will contain the number of increments left to move.

A special class B command exists to disarm an interrupt. In addition to interrupt commands, other class B commands exist for utility purposes such as overriding the wheel brakes when a bump interrupt occurs and for determining the status of certain interrupts and motor positions.

Operation of the TV preprocessor, when completed, will also be through special immediate-action level programs.

VII EXECUTIVE ROUTINES

In this section, we shall discuss the high-level programs that are initiated by an experimenter at a teletypewriter console when he commands the automaton to perform some mission. At present, a variety of FORTRAN programs exist that command the automaton to go from its present position to a certain designated position. These are called GOTO programs. GOTO1, GOTO2, and GOTO3 are three versions of increasing power and complexity.

Plans for the near future include implementation of executive routines. Such routines will make it possible to command the automaton to find an object of a designated type (FIND) and also to push a specified object from one place to a designated place (PUSH). It is important to note that FIND and PUSH should not necessarily be considered as being on a higher level than GOTO. While FIND and PUSH will involve calls to GOTO, it is also planned that GOTO routines will be written that will involve calls to FIND and PUSH as subroutines. At the end of this section, we shall outline preliminary specifications for such a planned "higher-level" GOTO program. First, we shall describe the GOTO programs currently being written.

A. GOTO1

GOTO1 implements a very crude strategy similar to that used earlier in the simulation GOODBUG (see the First Interim Report, p. 38). This program, which is currently operational, does not make use of the visual system or the model. It accepts as arguments the present automaton position and heading as well as the goal position. It then computes the heading and straight-line distance to the goal, and starts along that path. If its travel is interrupted by bumping into an object, it uses the simple expedient of calling a routine that turns it to the right and detours it around the object. Occasionally touching the object to determine its extent, the vehicle continues until it can again head straight for the goal. At such a time, GOTO1 is started from the beginning, and the process starts over.

GOTO1 will be used mainly as a way of becoming familiar with the operational characteristics of the vehicle and the immediate-action routines.

B. GOTO2

GOTO2 is a routine, now being debugged, that uses the touch and range-finder sensors to build a grid model but does not use vision. It can be called after the initialization routines have loaded a grid model that either partially or completely describes the automaton's surroundings. GOTO2 accepts as input the automaton's present position and heading and the goal position. It calls PLAN to find a proposed journey and then calls DOJOURNEY for execution. It provides PLAN with a parameter that defines the relative desirability of traversing unknown areas.

If GOTO2 receives a bump interrupt during execution of DOJOURNEY, it adds the newly discovered obstacle to the grid model and then immediately calls PLAN again. PLAN then proposes a new journey based on the updated model. DOJOURNEY is then called again, and so on, until the goal is reached. GOTO2 is very much like the simulation program written in LISP by Nilsson and Raphael.[7]

C. GOTO3

GOTO3 will use the visual system and accept as input the goal position. (The automaton's present position and heading with respect to various objects and walls in its environment is assumed to be established during a previously called initialization routine.) GOTO3 first calls PLAN for a proposed journey. It specifies the desirability of traversing unknown regions. After receiving a proposed journey, GOTO3 checks to determine whether any leg crosses an unknown region. Suppose the ith leg does. DOJOURNEY is then called to execute the first i-1 legs and any turn needed at the beginning of the ith leg. GOTO3 then calls the visual system to provide information about the region to be traversed by the ith leg. The model is updated by the new information thus received, and GOTO3 checks the updated model to see if the ith leg crosses a known empty region. If so, GOTO3 determines the next leg that crosses an

unknown region, and the process continues. On the other hand, if the ith leg is now seen to pass through objects, PLAN is called again before continuing the process.

GOTO3 may also have provisions for recalling certain initialization routines after having traversed a certain distance, so that the automaton's position with respect to its surroundings can be re-established. These routines use the visual system and range finder to sight walls and landmarks. If little error has accumulated, GOTO3 continues with the planned journey. If the error in positions and/or heading is large, PLAN is called again. If unexpected bumps ever occur, the position establishing initialization routines are automatically called, and PLAN is called again.

D. Preliminary Specifications for More Complex Programs

In this section, we shall outline as an example of a higher-level program a proposed method to solve a problem involving going to a position that is only implicitly defined, such as GO TO A DOORWAY. The highest levels of this problem-solving program will be written in LISP.

We assume that line, point, status, etc., information is kept in the grid and line models already described. In addition, the LISP programs will maintain at least the following "model" information: An object is represented by a property list, containing one or more attribute-value pairs. Typical attributes are CATEGORY, LOCATION, ORIENTATION, SIZE (e.g., approximate radius of circumscribed cylinder or sphere), COLOR, etc. A master list called OBJECTS contains all these property lists, grouped into sublists and indexed by object category. For example, the following list structure represents an environment with a cone, two cubes, and a doorway:

```
OBJECTS → CONE
           ((LOCATION (75 143) TYPE CONE SIZE 23))
          CUBE
           ((TYPE CUBE LOCATION (152 28) COLOR RED)
            (TYPE CUBE ORIENTATION 17 SIZE 82
             LOCATION (70 31)) )

          DOORWAY
           ((TYPE DOORWAY LOCATION (305 0) ORIENTATION 12
             SIZE 68)) ).
```

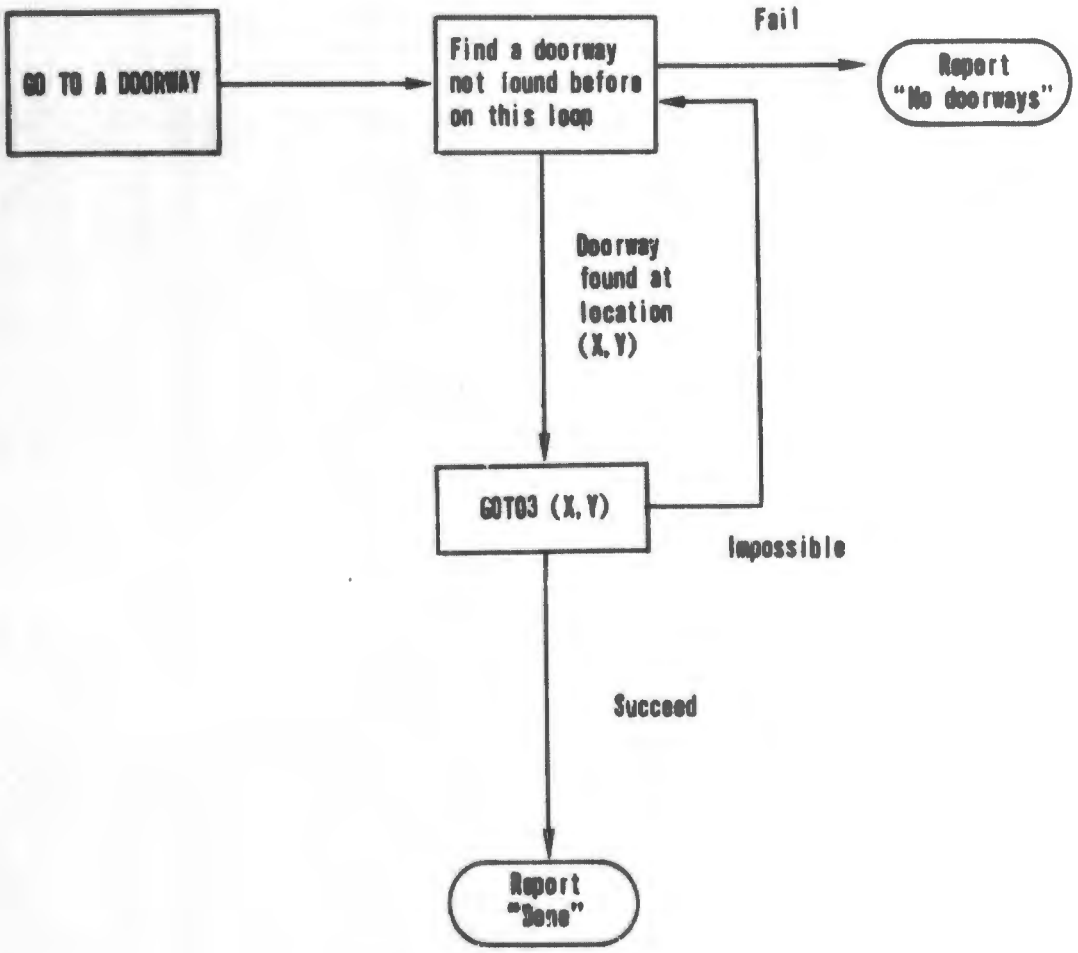
In addition, LISP will keep its own version of the grid model represented by a LISP array. Each cell of the array will contain either one of the atoms EMPTY or UNKNOWN, or a list of the property lists describing the objects known to be occupying the corresponding cell.

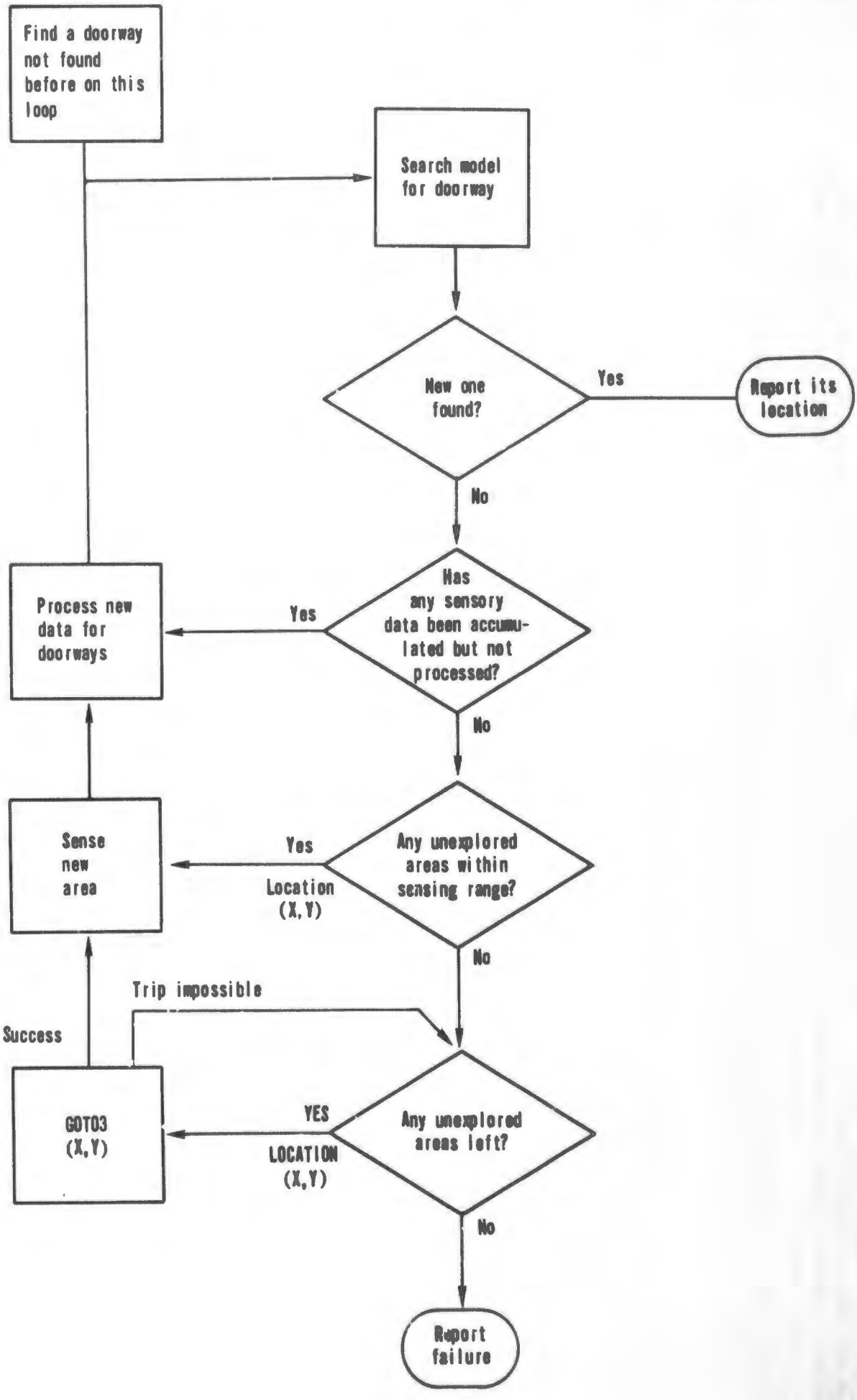
The problem we shall consider here is:

GO TO A DOORWAY.

Extensions of this program to handle problems of the form GO TO AN X, where X is any identifiable object, will be obvious. We shall probably write several programs for classes of problems at this level of complexity, and plan on eventually reducing more complicated problems to this level with Problem Analysis Programs.

The program will be described by a series of annotated, nested flow charts, starting from the top level.




Find a doorway not found before on this loop
Search model for doorway
New one found?
Yes
Report its location
No
Has any sensory data been accumulated but not processed?
Yes
Process new data for doorways
No
Any unexplored areas within sensing range?
Yes
Location (X,Y)
Sense new area
No
Trip impossible
Success
Any unexplored areas left?
YES
LOCATION (X,Y)
GOTO3 (X,Y)
No
Report failure

This is either the GOTO3 program described above for planning and executing a journey or a later version GOTO3.

Search the objects list. If it contains any doorways, return the location of the first new one and flag it as used in this problem.

Test the doorway-processed flag.

Call a FORTRAN subprogram that processes the grid and line models with appropriate tests to discover doorways. This program creates a sequential file of doorway locations, orientations, and widths, of the form

X_1 Y_1 θ_1 width

X_2 Y_2 θ_2 width

.
.
.

and returns as its LISP value the name of the file. LISP then reads the file and updates its OBJECTS list and grid-array model. It then updates flags and deletes the doorway information file created by the FORTRAN subprogram.

Look up current position on the STATE constants, compute "nearby" cell numbers, and look up grid model for UNKNOWN cells. If one is found, face toward it.

Search grid model.

There may be several "sensing" programs. In this case, we want a program that will take in from the real world, and add to the grid and line models, at least that raw data that the "search model for doorway" program needs. It also should provide information necessary for LISP to update its version of the model.

E. Preliminary Plans for a Specification Language

One desideratum for the specification language (in which commands will be given to the automaton) is that it be as close to natural language as possible. The reason for this is that we conceive of the task of communication with the automaton as an important part of the overall project, and many applications of automata would be facilitated by an English-like command language. Furthermore, a natural specification language would be easier for the experimenters to use during this project. The automaton described in this report knows a parametric universe [i.e., positions in terms of (X,Y) coordinates] that is alien to a person controlling the automaton, and that is also difficult to use. For example, a person might want the automaton to bring him a wastebasket, and he doesn't care where it is. The automaton must know that he is to go to (17,23), say, and that he will find a wastebasket there. Thus, it is crucial that the automaton have a scheme for translating natural language commands (natural to people) into simple commands compatible with its parametric universe (natural to it).

In this section we shall describe some early ideas about a possible command language that encompasses three sublanguages:

(1) A language for directives. Examples:

a.
```
START;
GO TO THE LARGE BOX ON YOUR LEFT;
TURN RIGHT;
STOP.
```

b.
```
START;
GO TO THE SMALL WEDGE;
BRING IT HERE;
STOP.
```

(2) A language for informing the automaton about his universe. Examples:

a. THERE IS A LARGE BOX 10 FEET TO YOUR LEFT.

b. THE TABLE HAS BEEN MOVED NEAR THE DOOR.

c. THE LARGE BLACK WEDGE HAS BEEN REMOVED.

(3) A query-language to ascertain the automaton's knowledge of its world. Examples:

a. DESCRIBE THE BOX ON YOUR LEFT.

b. WHERE IS THERE A LARGE WEDGE?

c. HOW MANY CUBES ARE THERE?

Each of these sublanguages deals with a sufficiently small subset of natural English to make implementation possible with known techniques, and yet is sufficiently large to permit rich manipulation of the automaton's universe.

The directive language has some special features that will enable it to name places and handle implicitly described places. Consider the following sequence of directives:

START
GO TO THE BOX 10 FEET TO YOUR RIGHT;
CALL THAT SPOT HOMEBASE;
MOVE IT HERE;
RETURN TO HOMEBASE;
STOP.

If we want the automaton to return to places where objects were (but are no longer), it is most convenient to name the specific place and keep a record of named places and their coordinates. The last object named (or located) is kept in the IT-STACK, and its coordinate position is automatically substituted for the position of any object implied by the presence of IT in a program directive.

One interpretive problem that crops up immediately is the "information match" between the problem statement and the modeled state of the world. For example, GO TO THE BOX 10 FEET TO YOUR LEFT may specify a unique box in the model, as well as providing its (X,Y) coordinates. (Note that the automaton's bearing must be stored so it can determine left-righters in the absence of good pattern recognition.) However, GO TO A BOX ON YOUR LEFT will be ambiguous if there is more than one box to the left of the automaton. This directive may be clarified by internal criteria (e.g., the nearest object that satisfies the conditions) or interactive

clarification. In the latter case, the automaton responds with a list of all objects that satisfy the specified criteria, and the user types in the name of the appropriate object. Lastly, tolerable vagueness must be resolved. Where is near the door? Since many human directives are vaguely specified in terms of across the room, near you, etc., each of these idiomatic specifications will be associated with an internal predicate to render an appropriate interpretation.

Appendix A

MECHANICAL DEVELOPMENT OF THE AUTOMATON VEHICLE

by

Vladimir Lieskovsky

Appendix A

MECHANICAL DEVELOPMENT OF THE AUTOMATON VEHICLE

A. General Arrangement of the Vehicle

At the beginning of the project, only very sketchy information was available about specific requirements for the vehicle. The general requirements given were that the vehicle should be able to maneuver on a linoleum-tiled laboratory floor, move on ramps that had up to a ten percent slope, be not wider than a doorway, weigh not more than approximately 200 lbs, move under radio-transmitted digital-computer control, and be energized by an on-board power source. It was further specified that the vehicle should be able to turn around its own vertical centerline in either direction and be able to move both forward and backward.

Accordingly, with this prescription we began with a rectangular platform, 3 ft in length and 2 ft in width, with the corners cut off at an angle. The platform was equipped with four wheels mounted in a diamond pattern: two 8-in diameter rubber castor wheels, one in front of the platform and one at the back; and two 8-in diameter rubber wheels, coaxially mounted, one at either side of the platform. The coaxially-mounted wheels were to be driven independently. One of the castor wheels was mounted on a spring-loaded flange, which allowed that wheel to deflect, under load, out of the plane determined by the other three wheels. In this way we achieved the compliance necessary to negotiate slopes. The platform stands about 10 inches above the floor level. The space provided between the wheels accommodates the main drive motors, and for a low center of gravity, the batteries.

A 4-in vertical distance above the platform was reserved for manipulator arms (described in Appendix B of Ref. 2). A standard 19-in electronic rack, supported at three points, was located above this reserved space. A video camera and range finder combination was mounted atop the rack.

B. Details of the Physical Arrangement

1. Power Supply and Drive

One of the first decisions to be made was the selection of the form of energy to be used for drive purposes. Among those considered were hydraulic, pneumatic, and eventually, electric drives. Since electrical power had to be made available for the electronics, electric drive was ultimately selected. The choice between secondary batteries and fuel cells was dictated mainly by price and delivery figures in favor of the batteries. Two 12-volt batteries in series were used to establish the operational, nominal voltage at 24 Vdc. The choice between drive motors was reduced to either a straight dc motor, an inverter and ac motor combination, or stepping motors. Complexity and control considerations of the digital commands ruled out the inverter/ac combination. Direct current motors, although electrically noisy, were attractive due to their high power density and good torque characteristics. Manufacturer's quotes were uniformly forbidding: six months for delivery and a price in excess of several thousand dollars for each motor. The units would have had standard clutches, brakes, and position readout capability for feedback information. Stepping motors, although they suffer from low power density, are excellently suited for digital control, and they were immediately available and were low in price (not more than about $200 each). Therefore the decision was made to use stepping motors exclusively for prime movers. Not all of the motors selected were rated at 24 Vdc, but they were easily converted by using dropping resistors.

In order not to lose count of the steps in the drive train between the motor and the drive wheel, the speed reduction between the motor and the wheels had to be one without slippage, that is, positive. The reduction was necessary to increase available torque from the motors and to reduce the amount of translation per incremental step of the motor to 1/32nd of an inch measured at the periphery of the wheel. For every control pulse, the stepping motor executes a rapid change in its angular position. Depending on the inertia of the driven load and the damping of the drive trains, oscillations may develop. These oscillations were

reduced by limiting the incremental stepsize, i.e., the generated amplitude. A cogged belt, or timing belt, arrangement was selected for the drive train. This was to give the necessary positive drive, while also introducing damping. As it turned out, the belt proved to be a secondary source of oscillations, since bending vibrations were generated in the belt when the stepping motor was operated. Increasing the belt tension reduced the oscillations to an acceptable level.

2. Closing the Minor Loop Through the Motor

The stepping motor operates in an open loop mode. Completion of any step depends on the inertial load coupled to the motor, and not unlike a synchronous motor, the stepping motor also can "fall out of phase," so to say, when it is overloaded. This condition is largely a function of the stepping rate. Therefore closing the loop in the operation of the main drive motors seemed to be warranted. Fortunately, similar considerations led Fredrikson[8] to synthesize, build, and describe a closed-loop stepping motor scheme. By using his results, we were able to adhere to the ground rule of no novel detail development. We closed the minor loop through the motor in the following way: a disk, containing fifty appropriate holes on a circle, was mounted on the motor shaft. Four light source and photocell pairs placed along that circle, and shifted by one-fourth of the hole pattern pitch, were mounted on the motor housing. This arrangement provided for 200 positions for every revolution, which is also the step-pattern of the motor. We used the simple schematic, described in Ref. 8 to complete the feedback loop. In operation, no step command can be given until after the information from the position feedback disk indicates that the previous step has been completed. Simply, the motor cannot miss a step.

3. Wheels

The rubber wheels presented another problem: due to their finite elasticity, transient motions generated either by the vehicle itself, or by its environment, resulted in disturbing oscillations of the whole vehicle in pitch and roll modes with a time constant of about 2 seconds.

This amount of settling time was judged to be unacceptable because no picture taking with the TV camera could be initiated during that time. Since friction on the driving wheels had to be maintained, but elasticity minimized, a properly-stiffened rubber driving rim on a metal wheel proved to be an acceptable solution. Since the castor wheels, however, could remain relatively compliant, but required reduced friction on the floor, they were capped with a metallic rim and gave good results.

The originally configured, independently-suspended castor wheel design gave way to a scheme that provided easy handling of the batteries. The supply batteries are now contained in a subcarriage, supported at three points. At one end of the subcarriage, one ball-bearing is located at each of the two corners while at the other end is located the vehicle's previously independently-suspended castor wheel. The batteries in the subcarriage can be conveniently wheeled to and from a recharging station. When the subcarriage is wheeled back to the vehicle, the ball-bearings are received by corresponding ramps, which lift up the ball-bearings and lock them into proper position. The bearings now act as pivots around which the subcarriage swings in a vertical plane. This freedom of movement provides for independent suspension of one of the four wheels. The distribution of the load on the vehicle is such that when the subcarriage is removed, the rest of the vehicle is still statically stable on its remaining three wheels.

4. TV Camera and Range Finder Mount

Although it is possible to scan with a TV camera which is rigidly mounted on a vehicle that is capable of turning around its own vertical axis, it seemed expedient to provide for an independent panning capability. Thus, the TV-range finder combination is mounted on a yoke that can be rotated by a vertically-mounted stepping motor. The yoke accommodates a transverse, horizontal axis, around which the TV camera can be tilted. The tilt drive train incorporates a worm drive and another stepping motor. The worm drive is necessary to cope with the excessive tipping moments originating from a revised version of the range finder. When the stepping motor is not in operation, the worm drive provides a

self-locking feature as an added bonus. In the pan mode, limit switches and stops are provided as well as an electromagnetic detent, acting on a 200-tooth gear, mounted on the shaft of a 200-step/revolution stepping motor. The yoke was designed for these functions only. The shaft of the pan motor is coaxially mounted with the vertical centerline of the vehicle; that is, if equal and opposite commands are given to the driven wheels, the location of the pan motor shaft does not change. The TV camera is located in such a fashion that the photosensitive surface of its vidicon tube is exactly at the intersection of the vertical pan axis and the tilt axis. Turning the vehicle about its vertical axis, panning the camera, and tilting it, does not affect the location of the vidicon surface, only its direction.

It also seemed expedient to attach the range finder directly to the TV camera. In this way, the distance of an object, viewed by the optical centerline of the TV camera, from the range finder can be measured.

A separate arrangement of the TV camera and the range finder was similarly logical: distance-mapping of the surroundings could be accomplished while the TV camera could "digest" and recognize a particular scene. However, the kinematic complexity of this arrangement seemed prohibitive when compared to the possible advantages.

Stepping motors were mounted onto the TV camera lens housing for computer controlled adjustment of the focus and the iris. Since these motors operate in the open loop mode, step count may be lost. Therefore separate limit switches for both focus and iris functions and at both ends of their range are provided. Whenever the limit switches are actuated, the counters are reset accordingly. This is also the scheme utilized in the pan and tilt modes.

5. Overhead Cabling

When it became clear that fund allocation temporarily ruled out the radio link with the central computer, an overhead suspended cable had to be used to supply power and communication to the vehicle. From a central location in the ceiling of the environmental room, the cable is connected

to the vehicle at its highest point to avoid obstruction of the view of the TV camera and range finder. The yoke was therefore capped to accommodate the cable.

Keeping 36 coaxial cables, each of 0.1-inch O.D., from tangling and kinking proved to be a sizable problem. Experiments with sheathing the cable with metallic, woven fabric turned out to be unsatisfactory due to excessive weight. Finally, cross-braiding was tried and accepted. The braids are held together with electric tape every foot or so. A trough, located in the false ceiling of the laboratory, through which a pulley travels acts as a reservoir for the unspent portion of the cable.

6. Tactile Sensors

Tactile sensors are mounted at the front and back and on both sides of the vehicle to provide protection against damage to the vehicle and to its surroundings and to provide touch information. These sensors were selected from commercially available microswitches, and are actuated by a flexible coil spring approximately 6 inches long. Piano wire whiskers or extensions may be added to the end of the coil springs to provide longer reach. The guiding principle has been to sense the presence of a solid object within the braking distance of the vehicle when it is traveling at top speed. Additional appropriately placed sensors protect the TV camera against collision in the translational and the rotational modes. The actuation of any sensor will inhibit the corresponding action, while override is also made available.

As further protection against collisions, heavy rubber bumperstrips are mounted on all protruding edges of the vehicle. If the performance capacity of the main drive motors permits, these bumpers will be used to move objects around the environmental room.

Appendix B

OPERATIONAL FEATURES OF THE AUTOMATON CONSOLE

by

Elmer Shapiro

Appendix B

OPERATIONAL FEATURES OF THE AUTOMATON CONSOLE

A. Introduction

The automaton console (see photograph in Fig. 19) is a collection of switches, lights, and logic. Through the use of the console, a human operator can directly (i.e., not employing the computer) issue commands to the vehicle and monitor the logical state of the vehicle. Essentially, the console permits the operator to execute the same vehicle operations executable by the computer. The logic associated with the console automatically permits time-shared access to the vehicle for both the console and the computer.

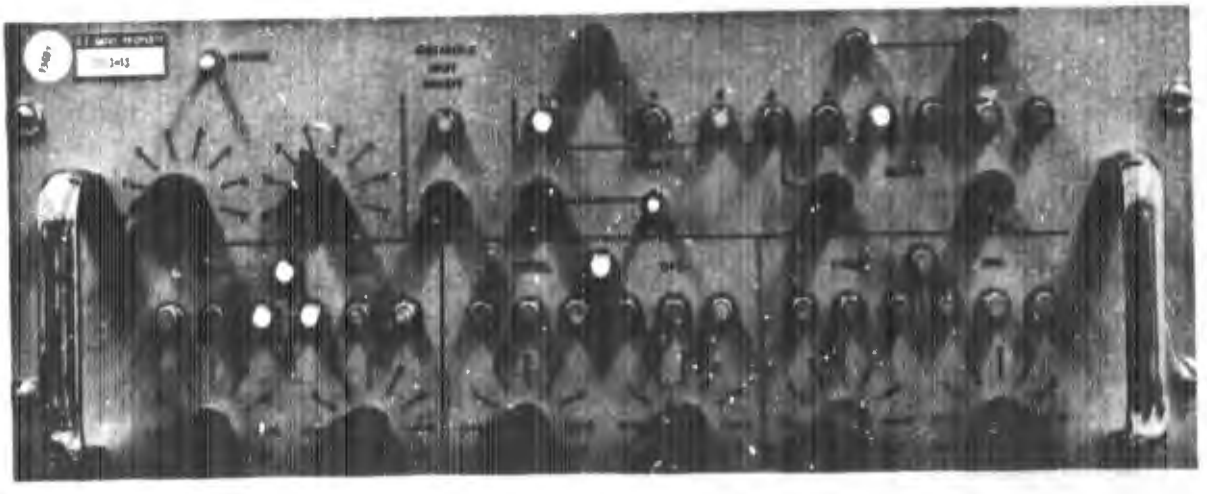


FIGURE 19. THE AUTOMATON CONSOLE

When an access conflict, in time, occurs between the console and the computer, the latter is given priority. When the console has been given access, however, the computer program is not given notice of the event, only a delay in Data Access Communication Channel (DACC) input/output occurs.

The intended purposes of the console are (1) to aid in debugging the vehicle hardware, (2) to assist in isolating system hardware faults, and (3) to assist in program debugging.

B. Writing Operations

The eight rotary switches of the console are used to specify information to be sent to the vehicle. This information represents a vehicle command. Each of the two switches labeled "MODULE ADDRESS" are used to specify the action logic module (ALM) on board the vehicle to respond to the command. All ALM addresses consist of two octal digits, hence each switch position can assume one of eight positions marked 0 through 7 inclusive. The leftmost switch represents the more significant octal digit of the address.

The data associated with the commands may consist of one, two, or three pairs of octal digits. The number of octal digit pairs that must be specified is uniquely determined by the particular ALM being addressed. These octal digits are specified by the settings on the bottom set of six rotary switches. Each octal digit pair is also labeled either MOST SIG, INTER SIG, or LEAST SIG--indicating the relative significance of each pair. Within each pair, the leftmost rotary switch is associated with the more significant octal digit.

Each switch can assume one of nine positions--labeled OFF, and 0 through 7 inclusive. When the ALM being addressed requires the use of three octal digit pairs, then none of the data switches should be in the OFF position. When the ALM requires the use of two octal digit pairs, then at least one of the LEAST SIG switches should be in the OFF position. As long as one switch is OFF the other may assume any position, including OFF. When the ALM requires only one octal digit pair, then at least one of the INTER SIG switches should be in the OFF position. Under these conditions, the position of either switch of the pair of the LEAST SIG switches is unimportant. The MOST SIG switches are set to the OFF position only for hardware test purposes.

Once the rotary switches are set, the information for one and only one command is sent to the vehicle by depressing and releasing the

push-button labeled WRITE. Continuous depression of the WRITE button will not cause any commands to be issued by the console. While the button is released, the operator has complete freedom regarding positioning or repositioning of the rotary switches.

C. Reading Operations

To read the information on board the vehicle and associated with an ALM, it is only necessary to set the address of the ALM of interest on the MODULE ADDRESS rotary switches and then depress and release the READ push-button. The information read from the vehicle will be displayed on the lights immediately above the module data rotary switches. The number of octal digit pairs of information read is dependent upon the module addressed. Associated with each set of six lights of an octal pair is a seventh light located above and between the middle of the set of six lights. When three octal digit pairs have been read, then all "seventh" lights will be lit. If two octal digit pairs have been read, then only the MOST SIG and INTER SIG seventh lights will be lit. The other six lights will be dark if the seventh light of the set is dark.

A lit lamp (of the set of six) indicates that the received bit had the value 1. A dark lamp indicates a value 0. The significance of the leftmost lamp is the greatest.

The settings of the module data switches do not affect the reading operation.

Each depression-release sequence of operations of the READ pushbutton causes only one read operation to occur, provided the toggle switch immediately to the right of the button is in the SINGLE position. If that switch is in the CONT (continuous) position, then an endless sequence of read operations occurs, without operator intervention. For the console, the read operation has priority over the write operation. To issue a command to the vehicle from the console, the read switch must be in the SINGLE position.

The read lights change state only when a new console read operation occurs. Computer operations or a console write operation will not affect the state of the lights.

D. Console Busy

The CONSOLE BUSY light is normally dark. It is lit only when the console is in communication with the vehicle or attempting to establish communication. If the computer is not communicating with the vehicle, then a single console read or write operation will busy the console for only a few milliseconds. When the console is set to CONT READ, then the console will be continuously busy. If the computer has access to the vehicle and a console read or write operation has been initiated, then the console will be busy while it awaits its turn.

Under some error conditions (see the next section) the console will be unable to complete a read or write operation, even when it has access to the vehicle. In this case the console remains busy until a reset is effected.

E. Interrupts and Errors

A horizontal line of nine lights, labeled 1 to 9 inclusive, at the top of the console, provide interrupt and error indications. These indicators can be turned on by either computer or console-initiated actions.

Lights 1 through 6 are associated with vehicle generated interrupts. The indications, by light number, mean, for light number:

(1) An "end of all action" interrupt occurred,
(2) An "emergency" interrupt occurred,
(3) A "bump" interrupt occurred,
(4) A "limit" interrupt occurred, and
(6) An "end of any action" interrupt occurred.

Light 5 is a spare, unused at present, and always dark.

Once an interrupt has occurred, the affected interrupt lights remain on until either the CONSOLE CLEAR or the MASTER CLEAR buttons are depressed. The former button affects only the interrupt lights, and

not the vehicle or computer. The MASTER CLEAR button (see the next section) does affect the vehicle.

Lights 7, 8, and 9 indicate data format errors detected by the vehicle. Normally, all these lights should be dark. If an error is detected, only one light will be lit, and the computer will have received an "FCU error" interrupt. These errors are "fatal," preventing the computer or the console from communicating commands or read requests with the vehicle. Recovery is possible only through a clear operation such as depressing the MASTER CLEAR button.

If light 7 is on, it is the result of sending too many data characters to an ALM by means of either a computer or a console-generated write operation.

If light 8 is on, it is the result of either the computer or the console attempting to obtain access to a nonresponding ALM, for either a read or a write operation. Access to a nonexistent ALM gives rise to such an error. Access to an ALM not "turned on" is another situation. Attempting to write into a read only ALM (e.g., the range finder) is another example.

If light 9 is on, it is the result of sending the vehicle an address or a control character when it expected a data character. Only a computer operation can cause such an error.

F. Control Operations

The console, FCU, and vehicle can be placed in their "normal" state from any "abnormal" state by the depression of the MASTER CLEAR button--EXCEPT, an emergency stop condition cannot be overcome this way.

In the normal state, the power control register on the vehicle is reset, thus turning off all vehicle power that is under program and console control. The vehicle is also cleared of any error conditions and is set to respond to a write or read operation. Any incomplete console read or write operation is cleared and the console is placed in the idle state. Any incomplete FCU operation is cleared and the FCU is placed in the idle state.

If the red EMERGENCY STOP pushbutton is depressed, the main (battery) power on the vehicle is shut off; also, a red light associated with the button is lit. The emergency stop condition remains in effect, preventing the reapplication of main vehicle power until the EMERGENCY STOP RESET pushbutton is depressed. The red light remains on while the emergency stop is in effect. Execution of the programmed emergency stop command produces the same effect as depressing the EMERGENCY STOP button. No programmed recovery is possible.

G. Turn on Procedures

The power to the fixed vehicle equipment must be turned on first. Typically (but not always) the emergency stop condition will be in effect and should be cleared. Then the MASTER CLEAR button should be depressed.

The main power on the vehicle should be turned on. This operation generally gives rise to spurious interrupts. These should be cleared by depressing either the CONSOLE CLEAR or the MASTER CLEAR buttons.

The vehicle and console are now ready. The computer need not be on, or if on, need not be programmed to cope with the vehicle, if only the console is to be used.

H. Power Modules

Note that, in order to operate the wheels, Power Module #2 must first be turned on. Pan and tilt require Module #3, and the range finder needs both Module #3 and #6. Module #4 will power the camera iris and focus.

REFERENCES

1. C. A. Rosen and N. J. Nilsson (Eds.), "Application of Intelligent Automata to Reconnaissance," First Interim Report, Contract AF30(602)-4147, RADC-TR-66-822, November 1966.

2. C. A. Rosen and N. J. Nilsson (Eds.), "Application of Intelligent Automata to Reconnaissance, Second Interim Report, Contract AF30(602)-4147, March 1967.

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

4. G. E. Forsen, "Processing Visual Data with an Automaton Eye," a paper presented at the Pattern Recognition Society Symposium on Automatic Photointerpretation, Washington, D.C., May 1967.

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

6. P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," submitted for publication, 1967.

7. N. Nilsson and B. Raphael, "Preliminary Design of an Intelligent Robot," Computer and Information Sciences-II (Academic Press Inc., New York, N.Y., 1967).

8. T. R. Fredrickson, "Closed Loop Stepping Motor Application," paper presented at the 3rd International Federation of Automatic Control Congress, London (June 20-25, 1966).

BLANK PAGE

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA · R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Stanford Research Institute Menlo Park, California 94025	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP N/A

3. REPORT TITLE

APPLICATION OF INTELLIGENT AUTOMATA TO RECONNAISSANCE

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Interim - 18 Mar 67 - 17 Dec 67

5. AUTHOR(S) (First name, middle initial, last name)
Edited by C. Rosen and N. Nilsson

6. REPORT DATE January 1968	7a. TOTAL NO. OF PAGES 54	7b. NO. OF REFS 8
8a. CONTRACT OR GRANT NO AF 30(602)-4147 b. PROJECT NO 4594 c. Task 459405 d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) RADC-TR-67-657	

10. DISTRIBUTION STATEMENT
This document is subject to special export controls and each transmittal to foreign governments or foreign nationals or their representatives may be made only with prior approval of RADC (EMIRC), Griffiss AFB, NY 13440.

11. SUPPLEMENTARY NOTES RADC Project Engineer: Capt. Stanley Saillie (EMIRC) AC 315-330-2146	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center (EMIRC) Griffiss Air Force Base, New York 13440

13. ABSTRACT

This report describes the results of research during the past nine months on the project, "Application of Intelligent Automata to Reconnaissance." The primary goal of this project is to investigate techniques in artificial intelligence applied to the control of a mobile automaton in a realistic environment. The main emphasis is on the design of a hierarchy of computer programs that will accept visual and other sensory information from the automaton and direct its actions toward the completion of missions requiring the abilities to plan ahead and to learn from previous experience.

DD FORM 1 NOV 65 1473

UNCLASSIFIED

Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Artificial Intelligence						
Pattern Recognition						
Computers						
Automata						
Modelling						
Decision Theory						