AD NUMBER: AD0820989
LIMITATION CHANGES
TO: Approved for public release; distribution is unlimited.
FROM: This document is subject to special export controls; 1 Sep 1967, and each transmittal to foreign governments, foreign nationals or representatives thereto may be made only with prior approval of RADC (EMLI), GAFB, NY 13440
AUTHORITY
ST-A USAF LTR, 17 SEP 1971

RADC-TR-67-171

APPLICATION OF INTELLIGENT AUTOMATA TO RECONNAISSANCE

Stanford Research Inst.

TECHNICAL REPORT NO. RADC-TR- 67-171

September 1967

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

APPLICATION OF INTELLIGENT AUTOMATA TO RECONNAISSANCE

Stanford Research Inst.

AFLC, GAFB, N.Y., 10 Oct 67-87

FOREWORD

This interim report was prepared by Charles A. Rosen and Nils J. Nilsson of Stanford Research Institute, Menlo Park, California, under Contract AF30(602)-4147, project number 4594, task number 459405. Reporting period covered November 1966 to March 1967. RADC project engineer is Captain Stanley F. Smillie (EMIRC).

This technical report has been reviewed by the Foreign Disclosure Policy Office (EML1). It is not releasable to the Clearinghouse for Federal Scientific and Technical Information because it contains information embargoed from release to Sino-Soviet Bloc Countries by AFR 400-10, "Strategic Trade Control Program."

This report has been reviewed and is approved.

Approved: Stanley F. Smillie

STANLEY F. SMILLIE
Captain, USAF
Interpretation & Analysis Section

Approved: William B. Moore

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

ABSTRACT

This report describes the results of research during the past four 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.

As of the beginning of March 1967, it is anticipated that the vehicle and other special hardware will be completed and be ready for preliminary experiments within two months. The preprocessor is expected to be completed a few weeks thereafter. Various "Immediate Action" computer programs are now being written to drive the registers on the vehicle and thus accomplish simple movements and sensing acts. In addition, maneuvering tactics designed during previous simulation experiments are being written into SDS 940 computer programs that will call the immediate action routines in the proper sequence to accomplish simple missions.

The SDS 940 computer system is now almost fully operational and will soon be utilized by the automaton project.

Our present status, then, can best be described as having nearly completed all preparatory work, placing us in a position to begin actual experiments with the automaton. We have arrived at this stage somewhat ahead of the schedule outlined at the beginning of the project; consequently, this opens the possibility of exerting a more intensive effort on programming the actual vehicle.

CONTENTS

CONTENTS (Continued)

CONTENTS (Concluded)

ILLUSTRATIONS

TABLES

I INTRODUCTION

A. Review of Progress

Work performed during the past four months on the Intelligent Automaton Project has brought us to a number of important milestones. This work is described in detail in the following sections of this report and is briefly summarized in this Introductory Section.*

In brief, the milestones are these:

(1) Completion of initial simulation studies of heuristics for automaton navigation and modeling of the environment.

(2) Turning up of the SDS 940 time-shared computing system.

(3) Design of a basic package of automaton software for the SDS 940. This software will control the automaton in simple navigational problems.

In addition, the automaton vehicle itself and the special visual preprocessing equipment will be almost complete and ready for use by the programmers in May 1967. It now appears that, during most of the second year of the project, we can experiment with getting the automaton vehicle to execute various tasks under direct control of the SDS 940.

Since we have reached this point somewhat ahead of schedule we can attempt to attack some fairly difficult programming problems, including more advanced problem-solving programs, during the present project. For project funds to be adequate to finance some of this extra programming, we have tentatively decided to stop fabrication of the pair of arms. (These are, however, designed.) We are also prepared to postpone the use of a radio link between the computer and the vehicle and use a cable connection, if absolutely necessary to preserve funds.

---

* For a complete description of the work performed during the first eight months of this project, as well as for a description of the goals and plans of the project, the reader is referred to the First Interim Report.

1. Status of the SDS 940 Computer

This system is now almost fully usable and will probably undergo acceptance tests within the next few weeks. Time-shared use of the computer is now being tried on an experimental basis.

2. Automaton Software

This system consists of a hierarchy of four levels. At the bottom level are the immediate action routines that can be used in FORTRAN programs to cause the vehicles to execute certain basic movements and sensor-reading acts. Next is the tactical level, which consists of programs calling immediate action routines in the proper sequence necessary to get the vehicle to execute simple navigational tasks. This level will probably make use of various stored models of the environment.

Next are strategic and problem analysis levels that will enable the execution of complex tasks requiring proper sequencing of navigational tactics. Most of the effort during the coming year will be devoted to writing programs at the immediate action and tactical levels, with the higher levels being worked on as remaining funds permit.

3. Navigation and Modeling

We have made some experimental designs of navigation tactics and modeling techniques. This work has resulted in several automaton simulation programs running on the CDC 3100. Efforts are now being made to transfer many of these programs over into the SDS 940 automaton software package at the tactical level.

4. Special Equipment

The vehicle, sensors, and other hardware are planned to be complete by the beginning of May 1967. The preprocessing hardware will probably be completed by June 1967.

5. Visual System Research

The goal of this research is for this system to provide the automaton with an efficient categorization of its immediate environment. One requirement is to determine the kind of environment the automaton finds itself in--e.g., Is it cluttered? Another important use is as an aid in navigation around objects or through passages. Of course, the basic function of the visual system is the discrimination between a number of types of simple objects in the automaton's field of view.

B. Organization of Work

The organization of the project into tasks for the coming year has been streamlined to reflect the changing nature of the work yet to be done. The following tasks remain:

1. Task 1--Visual System

The responsibilities of this task are:

(1) To design and write computer programs to operate the visual preprocessor under control of the automaton software system;

(2) To design and write computer programs to abstract information from the visual preprocessor useful for the automaton in the performance of its missions.

2. Task 3--Programming

The responsibility of this task is to design and implement the automaton software system necessary to meet the goals of the present project.

3. Task 5--Equipment

The responsibility of this task is to complete and check out all special equipment needed by the project. It is anticipated that this responsibility will be completely discharged within the next two or three months. At that time the task will be redefined to cover the responsibility of maintaining and augmenting this special equipment, as required.

4. Task 6--Basic Studies

This task was created to provide initial research on augmenting the capabilities of the automaton to include the ability to form simple generalizations, the ability to reason by analogy, and the ability to carry on a dialogue with a human operator. Work under this task has been proceeding at a low level of effort and there are no definite results to report yet.

5. Task 7--Program Management

The responsibility of this task continues to be managing and providing direction for the entire project.

The activities of the group previously designated as Task 2 have gradually shifted away from simulation studies of navigation tactics and modeling techniques and toward programming these features on the

SDS 940 for the actual automaton vehicle. Therefore, Task 2 has been dissolved as a separate group entity, and its personnel are now involved in the Task 3 programming work.

We have also decided to abolish Task 4, computer support, since the SDS 940 is now operational, and the automaton project will be one of the users of this facility. The facility will henceforth be separately managed.

II STATUS OF THE SDS 940 COMPUTER FACILITY

After numerous hardware and software difficulties, the SDS 940 computer is now almost fully operational. We are now operating in the time-sharing mode on an experimental basis with about ten simultaneous users, and installing on-line display software and special automata interface hardware. Major facility problems that are being satisfactorily resolved include the nature of the teletype interface, the use of the on-line CRT display, modification of the time-sharing monitor to handle the real-time requirements of the automaton, and improvements to various software subsystems.

A. Teletype Interface

We have adopted an interface developed by Bolt, Beranek & Newman, Inc., to permit an extremely flexible mix of local (directly connected) and remote (dataphone) users of the 940 computer. Each terminal line from the computer may go directly to a local teletype or--via a circuit card purchased from Bolt, Beranek & Newman--to a dataset for receiving calls from remote users. In addition, automata project workers may use the same local teletypes, circuit cards, and datasets to make calls to other time-shared computers, if desirable.

B. On-line Display

An on-line CRT display facility is useful for several aspects of this project, including:

(1) Simulation of the automaton's movements while developing behavioral strategies (as has been done on the CDC 3100 computer);

(2) Simulation of the video preprocessor for pattern-recognition studies;

(3) (Proposed) real-time display of aspects of the automaton's "thoughts"--e.g., its model of the environment--during actual experiments.

Our preparation for use of the display has proceeded through two stages:

(1) Preliminary programs for use of the display by a single user of the computer in 930 mode have been completed. These programs have already been used to simulate the preprocessor, producing much more effective and useful displays than the line printer previously used.

(2) Tentative specifications have been drawn up for FORTRAN-level control programs for time-shared use of the display. The proposed system would permit a 940 user to create a display consisting of points, lines, and text; identify and label arbitrary portions of the display; and use the light pen to manipulate or erase labeled items--all without significantly affecting other users of the time-sharing system. These specifications, which will soon be implemented, are attached as Appendix A to this report.

C. Time-Sharing Modifications

The automaton program will make some unusual demands upon the time-sharing system monitor and executive programs: the executive program must be able to control all input-output operations over the special direct access communication channel (DACC) associated with the automaton; the monitor program must be able to respond in real time to interrupt signals from the vehicle; and, during experiments, the automaton control programs must have priority over all other users of the time-sharing system. Detailed design of these changes to the standard SDS software is now in progress.

D. Software Subsystems

Recent developments in two software subsystems are of interest to the automaton project.

(1) A new version of FORTRAN has recently been delivered by SDS which permits the use of 940 assembly language subprograms. This facility is essential for convenient control of the automaton and the on-line display from FORTRAN.

(2) Bolt, Beranek & Newman are developing an extended LISP 1.5 programming system that uses drum memory for part of the free space list, has powerful on-line debugging aids, and operates under the 940 monitor. Preliminary versions are already available. This system may be of great value in the development of high-level strategies and models for the automaton.

III SPECIFICATIONS FOR THE AUTOMATON SOFTWARE

The specific automaton software will consist of four major hierarchical levels of programs, which we shall call the problem analysis, strategic, tactical, and immediate control levels. Programs at each level are expected to get their inputs from and report their results to a program at the next higher level, and to accomplish their jobs by executing programs at the next lower level. The problem analysis level will translate the original problem statement into a sequence of well-defined subgoals, each requiring a particular mode of overall automaton behavior; the strategic level program will select a suitable tactical method and monitor its effectiveness; each tactics level program executes a particular method for achieving a particular type of subgoal; and the immediate control level consists of drivers for automaton devices, preliminary sensory data processing, and interrupt handling programs.

A. Problem Analysis Level

We have considered various kinds of problems we would like the automaton to solve. These problems are usually stated in English, which of course is not yet acceptable input to a computer. Therefore the following three jobs will have to be done in order to supply the automaton system with a useful interface with the outside world.

1. Definition of a Problem Specification Language

This will be the language ultimately to be used at a control teletype to describe a problem to the machine. This language should be simple enough to be used by a novice, rich enough to describe a large, interesting class of problems, and limited so that any statable problem is at least potentially soluble by the machine.

2. Definition of the Automaton's Major Modes of Behavior

Any task that the automaton can perform may be described as a sequence of different actions or different kinds of activity. We must define a simple control language in which this procedural definition of a problem may be stated.

3. Implementation of a Specification Language to Mode Control Language Translator

The purpose of the problem analysis level is to translate from a human-oriented problem description into a machine-oriented description. Of course, this translation process cannot be implemented until both the source and target languages are precisely defined.

We have decided to concentrate our initial efforts on Item 2, the definition of major modes of behavior, since this mode control language must interact with the lower levels of the software structure. Items 1 and 3, design of a problem specification language and its translation into the mode language, will be left until later in this project.

Automaton modes of behavior may be classified into four major groups: action, information gathering, exploratory, and communication. The action modes govern the physical motion of the automaton and its appendages. These may include "GO TO," "PAUSE," and "GRASP." Information-gathering modes conduct directed searches for well-defined data. For example, the "FIND (OBJECT)" activity could attempt to determine the location of a specified object by just searching a "mental" model of the environment, then investigating new sensory data, and finally wandering about the environment by subsidiary use of the "GO TO" mode. (Thus we see that one type of mode may be used within another.)

The automaton will revert to an exploratory mode of behavior whenever it has nothing more important to do. Such exploration could consist of either aimless wandering or purposeful investigation to resolve any incomplete or ambiguous data in its memory.

In a communication mode the automaton suspends all action while it receives new information or answers questions via a teletype. Such modes will be useful initially for debugging, and eventually for doing tasks like, "Explore the back of the moon, and then come back and tell me about the interesting things you discovered."

Our first efforts will concentrate on the "GO TO" and "FIND" modes. Other modes will be added later as far as our resources permit.

B. Strategic Level

The main strategic program will be "interpreter" of the problem task as stated in the mode control language. It will have available one or more methods for achieving each subgoal, and applicability criteria for each method. It will also have "memory" of previous experience in the form of an updated model of the environment. The strategic program will also have to monitor the actions of its subprograms and decide how to respond to signals from the vehicle.

We are presently studying how to structure a model of the real environment and how the strategic program can best use such a model. (See Sec. IV-B of this report.)

C. Tactical Level

These are the programs that actually "do" something. For each subprogram type represented by a possible mode of automaton behavior, there must be one (or more) methods for attacking the associated problems.

Each specific method is embodied in a separate program. Each of these method programs may contain its own heuristic (or nonheuristic) problem-solving techniques, and its own special "models" of the environment. In addition, these programs may refer to, and help update, the basic model generally maintained at the strategic level.

The tactical method programs call for specific actions on the part of the vehicle (by means of the lower-level immediate-action programs); therefore, they must be responsible for requesting sensory data from the vehicle and initializing for appropriate actions should unexpected or emergency information arrive from the vehicle.

The first tactical programs to be implemented will be adaptations of the programs used in the automaton simulations on the Control Data 3100 computer. Some of these were described in the First Interim Report. Other than making minor changes in the source language, the main problem in transferring these FORTRAN programs to the SDS 940 computer is to replace subprograms that simulate sensors and motors with subprograms that control real devices. The interface between actual automaton hardware and tactical level FORTRAN programs has now been designed and is discussed as part of the Immediate Action level (below).

D. Immediate Action Level

In order to specify a software interface with the automaton vehicle, we must first establish what signals will be transmitted between the vehicle and the computer. The computer can send messages to the vehicle to activate various motors, and the vehicle can generate "program interrupts" to the computer when various conditions arise. The nature of the communications channel also affects the number of different kinds of signals that can be transmitted. We have now specified, for the cable-connected vehicle, what vehicle conditions will generate interrupts to the computer, what major commands the program can send to the vehicle, and how these interrupts and commands will be specified and controlled from tactical level FORTRAN programs.

1. Interrupts from Vehicle

The cable-connected vehicle will be able to issue five distinct signals to interrupt the computer's central processing unit.

(1) Limit switch interrupt, generated when any motor on the vehicle reaches a limit stop or clutch slip condition. The interrupt service program may then interrogate the motor status to determine which limit was reached.

(2) Bump interrupt, generated when any of the cat's whisker touch sensors is disturbed. The interrupt service program may then interrogate the cat's whisker groups to determine the point and, therefore, direction of contact.

(3) Emergency interrupt, generated by such conditions as the vehicle's overheating, falling over, etc.

(4) End of action interrupt, generated when any "action logic module" reaches a quiescent state--i.e., when any prescribed vehicle motion or activity reaches its normal completion. The program may always interrogate the vehicle to determine which action logic modules are busy and which are idle.

(5) End of all action interrupt, generated upon a transition from any action logic module busy, to all action logic modules quiescent. This is useful to notify the program that several actions performed in parallel are all completed.

When the radio communications link is added, additional interrupts will be used to reduce the amount of interrogation and decoding required for the program to establish the specific source of the interrupt signal.

2. Commands to the Vehicle

The principal commands the program can issue to the vehicle will cause the vehicle's wheels to move (forward or backward, independently or coupled), and the camera--together with its coupled range finder--to pan and tilt. In addition, the computer can read range, interrogate the status of various devices on the vehicle, and override certain hardware reflex actions on the vehicle (e.g., any "bump" sensor would cause the drive wheels to stop unless this reflex were overridden by the program.)

At the machine language programming level, the computer communicates with the vehicle over a DAC'' input/output system. We are now designing the interface between higher level tactical programs and the vehicle commands and interrupts.

3. Philosophy for FORTRAN-Level Vehicle Control

The commands proposed below will enable a FORTRAN program to "use" the automaton; in particular, to move it and to obtain range and touch data. Method of control of the visual system has not yet been specified, but will be substantially similar.

Three classes of FORTRAN subroutines will be available to control the automaton. Class A commands are the ones that actually cause logic modules (ALM) to perform. Class B commands either pass information to or receive information from an ALM. Class C commands are utility commands that are concerned with saving important data or coordinating actions.

A Class A command is generally issued by a FORTRAN statement of the form

```
CALL CMNDA (N, IHANDL, LABEL)
```

where the subroutine name CMNDA identifies an ALM to be activated and the three subroutine parameters have the following functions:

N is the magnitude of the desired action--e.g., the number of increments to move.

IHANDL is a flag used to indicate the status of the action.

The issuance of a Class A command merely initiates an action; the calling program then proceeds in parallel with the accomplishment of the action. This calling program may interrogate the state of the action by testing IHANDL, which will be equal to, less than, or greater than zero according to whether the action is in progress, successfully completed, or abnormally terminated, respectively.

LABEL is used to identify a particular instance of a command in future statements. For example, the Class C command

```
CALL   WAIT (MOVING)
```

would suspend program activity until completion of action that was initiated by a previous Class A command labeled MOVING, such as

```
CALL   MOVE (NSTEPS, IHANDL, MOVING)
```

Class B and C commands have formats similar to that of Class A commands, except that some of the parameters are unnecessary and therefore omitted.

Class A

(In all cases, bump or limit conditions may cause abnormal termination of the requested action.)

(1) CALL LEFT (N, IHANDL, IBOX)

(2) CALL RIGHT (N, IHANDL, IBOX)

These commands move the left (or right) drive wheels forward (N positive) or backward (N negative) N increments of 1/32nd of an inch.

(3) CALL MOVE (N, IHANDL, IBOX)

Move the vehicle (by moving both wheels simultaneously) N/32 inches.

(4) CALL TURN (N, IHANDL, IBOX)

Turn the vehicle abouts its vertical axis by moving both wheels simultaneously in opposite directions N/32 inches each.

(5) CALL PAN (N, IHANDL, IBOX)

(6) CALL TILT (N, IHANDL, IBOX)

Pan (or tilt) the camera/range-finder unit N increments of 1.8^{0}.

Class B

(1) CALL RANGE (N, IBOX).

N, initially zero, is set to the current range value.

(2) CALL OVRID (N)

Override the "stopping" reflex to a bump. The value of N specifies which wheel brake(s) to override.

(3) CALL CEASE (N IBOX)

Interrupt and stop current action of the Class A command initiated with a call that referenced the variable IBOX. N will contain the number of increments already accomplished by the Class A command.

Class C

The exact format of the Class C commands has not yet been specified. The purpose of most of these commands is to identify sections of FORTRAN code that should be executed upon the arrival of interrupt signals from the vehicle. Separate sections can be specified for each kind of interrupt and each condition under which an interrupt may occur. A special Class C command will permit these "interrupt service" sections of program to return control to the interrupted program.

IV DESIGN OF TACTICS AND MODELS

Work has continued on algorithms for model building and simulation of vehicle navigation strategies with the aim of creating prototype programs that could be transferred (more or less intact) from the status of navigation simulations on the CDC 3100 to real vehicle control programs on the SDS 940 system.

The following three specific problems received most of our attention:

(1) Modification of the programs so that the simulations were of an automaton of finite width;

(2) Development of routines and strategies for building a model of the environment; and

(3) Development of navigational tactics for using the model.

This section will consist of a discussion of progress in these areas.

In addition to the above activities, a 400-foot color movie was prepared illustrating the goal-seeking behavior of an "intelligent" automaton.* This film consists mainly of photographed sequences taken directly from the scope of the CDC 3100 display. These sequences depict the behavior of a simulated automaton in both discrete (cellular) and continuous environments of a variety of types. The journeys employ several different strategies involving different degrees of perceptual ability on the part of the "bug." The model-building routine is also illustrated.

A. Modification of the Simulation Programs

In all previous simulations the automaton was effectively of zero size. This led to situations where the automaton passed very close to some obstruction, or through a very small gap between two obstructions.

---

* This film has been presented at talks given by C. A. Rosen at the Conference for Intelligence and Intelligent Systems, University of Georgia, January 15, 1967, and by B. Raphael at the University of Michigan Communications Sciences Colloquium, February 7, 1967; it was shown again at a lecture by M. W. Green at Stanford University on February 23, 1967. Also, G. E. Forsen presented a paper entitled "Preprocessor for an Automaton's Eye," presented at the 1967 Winter Convention on Aerospace and Electronic Systems, February 7-9, 1967.

In order to make the simulation more realistic, two changes were made. First, straight line automaton paths were possible only when clear of obstructions for a finite distance on either side. Second, when a strategy called for the automaton to go close to a corner of an obstruction, the automaton was made to pass by the obstruction at a finite (and predetermined) distance, as shown in Fig. 1. In addition to solving the above problems, more meaningful data for the range finder (simulator) were obtained when objects were not approached too closely.

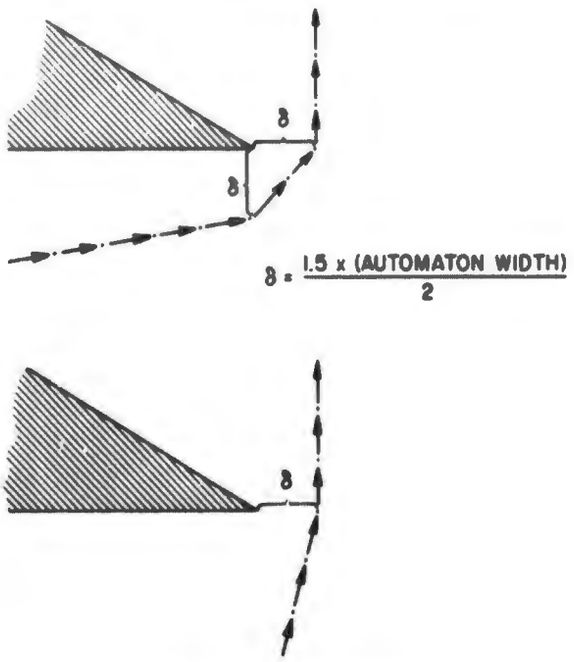


FIG. 1 EXAMPLES OF PASSING CLOSE TO CORNER USING FINITE WIDTH SIMULATION

One effect of the changes was to introduce situations whereby the goal (or sub-goal) could be seen but could not be reached along a straight line path. The navigational strategy was modified to handle this situation.

The investigation of maneuvering strategies has reached a point where further possible improvements are relatively trivial compared with the improvements that are possible if the automaton has available to it a model in which trial journeys can be made. All further work on this subject has assumed that such a modeling capability is included in the automaton system.

B. Modeling

1. Straight-Line Representations

A model would ideally include all that has ever been discovered about the environment. This, however, would involve a far larger storage system than is really necessary; therefore, some data compression should be carried out. The method used in the strategies so far tested is to use a number of straight line segments to represent the outline of objects in the environment. Section IV-B-3 discusses the problem of what should be modeled (objects, space, regions of uncertainty, etc.); here we are concerned only with the mechanisms of carrying out a meaningful data compression by building a model consisting of straight line segments.

The modeling to date has assumed that the primary sensing data is from the range finder. This device will have significant error characteristics--roughly 10% in range and a fraction of a degree in angle. In addition, it will have a limited range (up to about 30 feet). No information from the TV camera system was assumed. Figure 2 shows a sample environment and the readings that were assumed from a range finder whose principal source of error was truncation error.

A least-squares-fit subprogram was written that enabled a line to be fitted to a series of points. The modeling routine decided how many points to try to fit with one straight line. This was resolved by the following method.

Take all the readings in order until a reading is obtained where the range is significantly different from the previous one. This will break the whole set of readings into groups, each of which should be approximated by a single straight line or by a set of connected straight line segments. Within each group, progressively more points can be fitted to a single line until the fit obtained is not as good as should be obtained, taking into account the accuracy to be expected in the particular range readings. At this point a new line segment should be attempted for the remainder of the points.

After the automaton has taken some readings and built the first model, any successive readings must be merged with the old model in some way. The new readings may be more accurate or less accurate than the readings that were used when the model was built originally. As a measure of the accuracy of a particular part of a model we have defined a "confidence" figure for the end-points of a line in the model. This confidence is inversely proportional to the range of those readings from which the line in the model was constructed. It gives a way of estimating the probable error in the model. When a new reading is taken, the new information (with its confidence) can be combined with the old information and a new and higher confidence assigned to the modified line segment. The model will therefore grow as new information is obtained about the environment. The amount of storage required for the model will not become too large if certain methods are used to merge and remove lines from the model. These techniques are:

(1) Join a line to another if its end point is close to the other line. This reduces the number of end points that must be stored.

(2) Merge two line segments that overlap and are parallel.

(3) Remove all lines below a certain length.

Some examples of such operations upon the model are illustrated in Fig. 3 (not to scale).

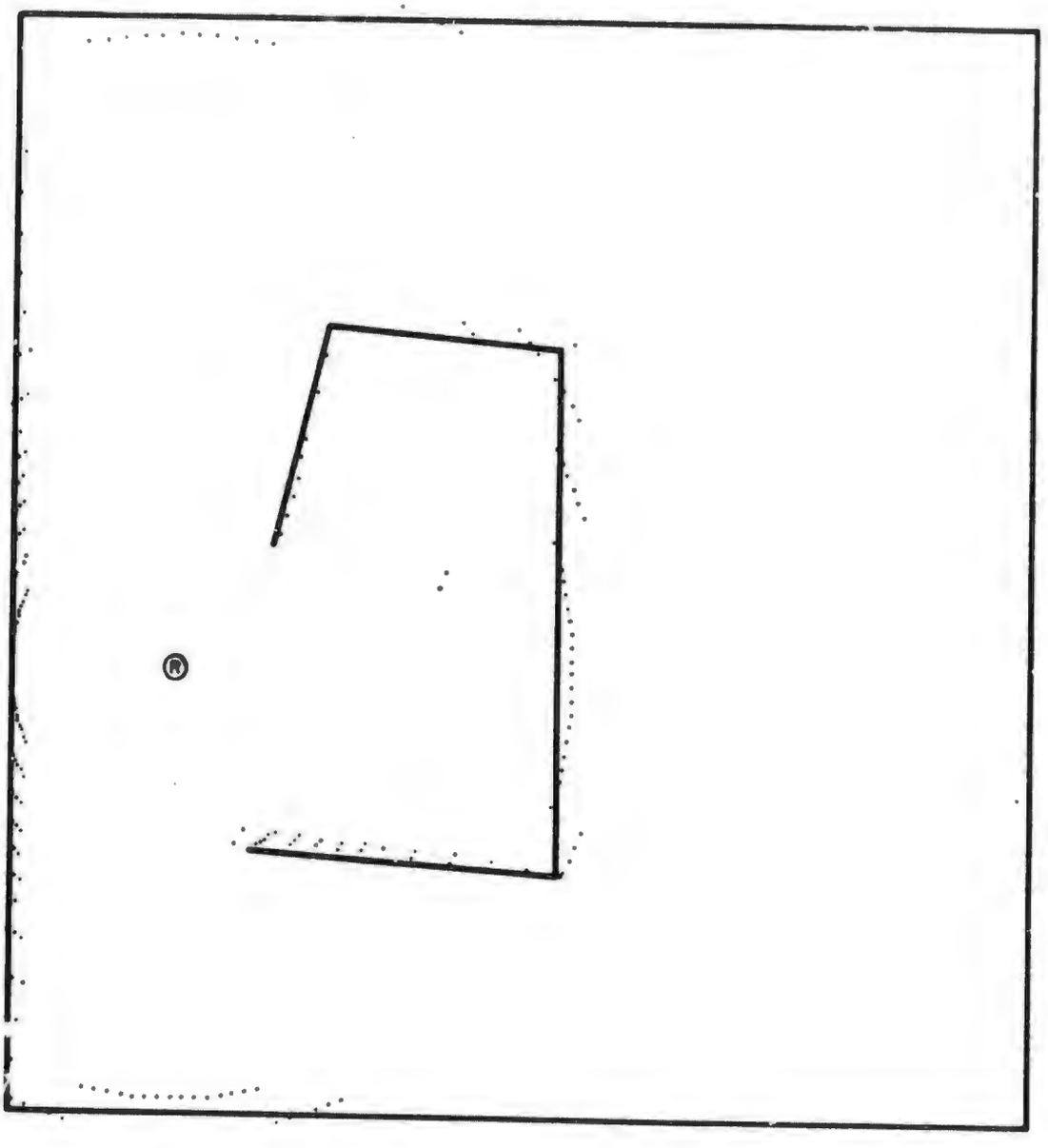


FIG. 2 AN ENVIRONMENT CONSISTING OF ROOM AND OBSTACLE (Solid Lines) AS SEEN BY A SIMULATED RANGE FINDER WITH TRUNCATION ERROR (Dotted Arcs)

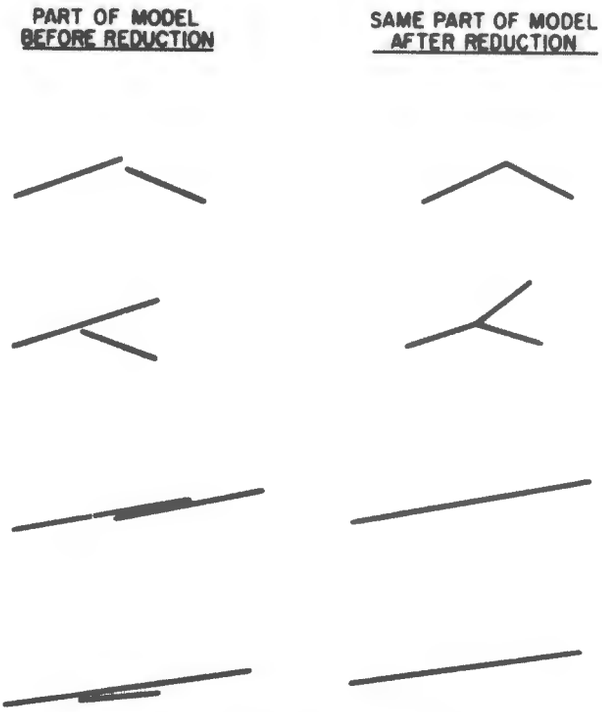


FIG. 3 WAYS IN WHICH THE MODEL CAN BE REFINED

A considerable amount of computation has to be carried out in building the model. This can be reduced if the method of building the model is changed. In the method described above, the readings of the range finder for a complete 360^0 sweep from each position occupied by the automaton, are used. We have started an examination of a different method, described below.

Initially the model of the automaton will be blank. The first action of the automaton will be to find an object that is close. As soon as this object has been found, an extensive examination of it will be made with all sensors available to the automaton. This examination will enable an accurate representation to be made in the automaton of this particular object. When the object has been studied enough, the automaton will concentrate on the next nearest object, and so on. At any time that this modeling is going on, the automaton may be given a task to perform. It will then break off its examination of the current object, and proceed about its task, using the best model that it has built to date. At the completion of the task the automaton will return to complete the modeling of the partially studied object. The automaton will not in general try to update its model with data that was obtained while carrying out some other task. This means that when modeling, the automaton will have to access only a small part of the data that makes up its model, thereby greatly increasing the speed of modeling. Uncertainties in constructing the model will not be resolved by sophisticated processes that attempt

to infer as much as possible from each measurement, but rather by taking more measurements that will resolve ambiguities.

2. Data and Program Structure

A list-like structure was developed whereby a line was defined as two pointers to a list of x, y coordinates. Thus, the coordinate pairs associated with a line were the end-points of the line. Each end-point had a confidence value associated with it.

Entities can be identified and maintained as lists of lines. Entities are such objects as chairs, tables, etc. The advantage of this method is to associate each of the entities with a particular area of the automaton's environment. This will lead to improvement in the run-time of certain of the subprograms, because all of the model will not need to be searched to answer some questions.

For the future, subprograms will need to be modified and a program structure developed to allow certain of the simulation programs to be more readily usable when merged into the actual running system. Three general types of subprograms now exist: housekeeping (e.g., debugging, display, initialization, etc.), hardware simulation (e.g., range finder), and tactics (e.g., "go to"). The programs now available for the CDC 3100 will be converted for the SDS 940. New subprograms will need to be developed for all three types. As the hardware simulation subprograms are converted, the names and parameter strings will be adjusted to conform with the hardware interface programs. Thus, when the actual system is running, the temporary absence of certain pieces of hardware may not cripple the program system because a simulation subprogram may be substituted. The tactic program names and parameter strings will be redefined for future use by the higher level strategy programs.

3. Suggested Conceptual Views as Basis for a Model

The world of the automaton should be viewed as dualistic. The two sides of the dualism consist of objects and voids (or occupied space and unoccupied space). The type of simple environment most often considered consists of a single, multiply-connected void surrounding a number of objects. The mere closing of boxes or of all openings in rooms creates multiple voids.

One reason for making a distinction between voids and objects (i.e., not just considering a void to be another kind of object) is that in distinguishing one void from another, there are no attributes of a nonmetric nature to consider. This statement may be viewed as a definition, but need not imply vacuum. Water might be void to an ocean-going automaton, but not to a land inhabitant. As for uniqueness of the definition, one might argue that there are no nonmetric attributes to distinguish one cube from another. The reply is that the cube's material properties are important, even if only implicitly. An automaton instructed to look for a cube should hunt for something solid, at least on the surface, not for the void inside a cubic box.

In its model of the world, the automaton should have access to previously acquired metric information on all important voids. It should retain similar metric information on objects, plus any number of representations of additional attributes of importance. In some cases, naming or identification of an object on a higher level (e.g., a Type-A Chair having a location and orientation, L) should replace detailed information in fast memory (dimensions or other details being relegated to lower-speed, back-up storage).

Let us now concentrate solely on the metric representation of objects and voids. Any programming of a model to date (discrete or continuous) has connoted retention of sensed metric information on objects but not on voids. In other words, there has been no programming for memory of open spaces, except whatever could be derived from memory of objects. But with such a model, there is obviously no way to distinguish between regions previously observed to be empty and regions not yet observed. The question is not one of precision of the model, but rather of wholesale discarding of an important portion of sensed information.

The defect indicated above may be removed by incorporating a representation of voids into the model. This could, of course, be done by viewing a void as just another kind of object, and thus retaining a unitary model. But it seems preferable to use a dualistic model (i.e., one having a primary division between the representations of voids as opposed to objects), in line with the above discussion of a dualistic interpretation of the automaton's world. In Fig. 4, hatched areas represent voids, and heavy lines represent objects. Surrounding blank areas represent the part of the environment not yet observed by the automaton. (The environment here is a tall, double bookcase in a dead-end passage.)

Note that a dualistic model implies a three-way splitting of the automaton's impression of the metric aspects of its world--namely, into the categories of (1) objects, (2) voids, and (3) the unknown. Obviously, substitution of a representation of the unknown regions for that of either voids or objects would allow for an equally complete metric modeling, since any one category is always the spatial complement of the other two, taken together. However, since nonmetric representations of objects must

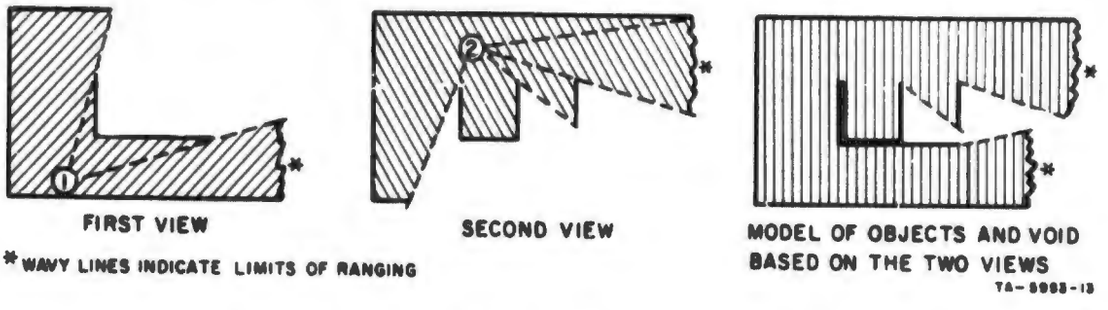


FIG. 4 EXAMPLES OF BUILDING A DUALISTIC MODEL

also be accumulated, this category should always be one of the two recorded in the model. In general, a representation of voids should be the other part of the model, because of the direct usefulness of known open spaces to the robot for navigation, for obtaining a mental image of known available space from a remote location, etc.

If it can be assumed that the robot's knowledge of its world would typically increase to the point where a record of the unknown would occupy substantially less memory than the record of voids, it might be worthwhile to program for substitution of the former record for the latter under certain conditions. This could be done by deriving a direct representation of the unknown from the initial dualistic model and then discarding the representation of voids (thus leaving a converted dualistic model of reduced size).

4. Expansion of Present Modeling to Dualistic Form

Before discussing some possible ways for building a model in three dimensions, let us consider how the present two-dimensional model might be augmented to dualistic form.

Although the work on a programmed model is in transition, the basic representation to date has been in terms of a list of line segments, with associated confidence levels. For our purposes here, it is unimportant whether a single long list is kept or whether--at a higher level, for explicit representation of individual objects--segments are arranged in groups or in a multilist structure. An important feature to note, however, is that a line tends to be "centrally" located among the set of range points it represents--such as by a least-mean-squares fit. Confidence levels are derived from "plus-or-minus" tolerances associated with the range data. For portrayal of such a fit, note Line AB in Fig. 5. The ellipses represent tolerances on the range data points.

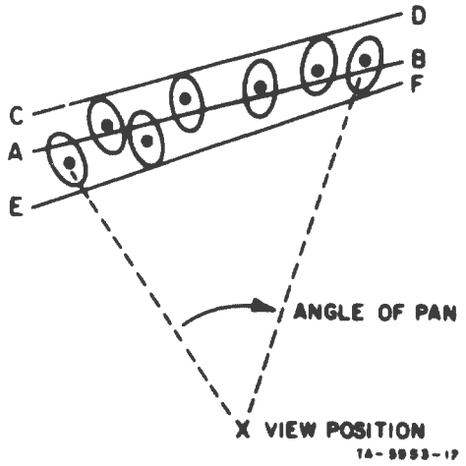


FIG. 5 FITTING OBJECT AND VOID BOUNDARIES TO RANGE DATA

The first possibility considered for a dualistic model would, if implemented, require a considerable change in the model-building procedures. To represent an object boundary, a line would be generated as an approximation to the upper tolerance limits of the range data (Line CD of Fig. 5). Similarly, a line associated with the lower tolerance limits (Line EF) would represent a boundary of the void area observed. The degree of confidence would be implicitly associated with the width of the band of obscurity between such a pair of lines. Further, with some specified low probability of error, one

could assume the object boundary to be at l ast as close (to current automaton position), and the void boundary to be at least as far away, as the respective approximating lines in the model.

It is apparent that there is no need to handle the modeling of objects and voids in exactly the same fashion. Rather, a different approach would be to carry over as much as possible of the current software for modeling objects, and to use the above method for modeling voids.

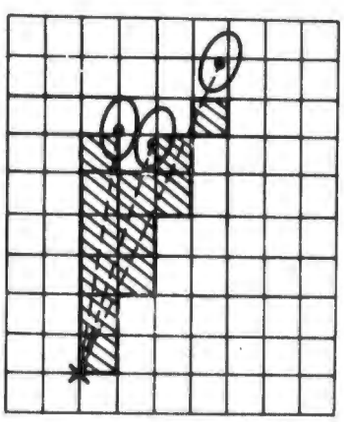
For still another possibility, consider that the purposes of having an explicit record of observed voids might be quite well satisfied by using a cruder representation for this portion of the model--say, just a square cellular structure. One way to set this up is to add a cell to a list of empty cells if it satisfies each of the following conditions:

(1) The cell is not already on the list.

(2) One or more lines of sight pass through the cell.

(3) The center of the cell is not covered by the area of uncertainty around any one of the endpoints of these intercepting lines of sight.

There would, of course, have to be some means for removing from the list cells later observed to be occupied. The sketch below is an example of applying the above procedure in one case of 3 range points.



TA-8983-11

5. Three-Dimensional Modeling

The last type of representation for voids mentioned above is extended to three dimensions merely by replacing the square cells by cubes (or perhaps parallelopipeds). The main question is one of how fine to make this cellular structure under conditions of limited storage space. If considerable precision is required, and there is a lot of empty space, it might be more efficient to return to a basically polygonal representation of voids. Or, sets of empty cells (e.g., a module 2 × 2 × 2) might be grouped into larger cells in regions away from boundaries.

Several types of structure have been considered for the three-dimensional modeling of objects. The discussion in this section is based on use of range data only, with no inputs assumed from the pattern recognition equipment.

In two dimensions, each range data point is derived from a line boundary between a planar object and a planar void. In three dimensions, each data point is derived from a surface separating an object from a void. Continuation of this analogy implies a polyhedral representation

of objects, corresponding to the polygonal modeling in two dimensions. But the implied procedure of associating space lines to form planes and then associating these planes into polyhedrons seems too cumbersome.

By another approach, we note that ranging with zero tilt amounts to sensing a contour line at the elevation of the ranging device. Ranging at other elevations would provide data for polygonal approximations to contour lines. Use of such a contour representation here is less convenient than in normal topographic mapping--in mining, in highway planning, etc.--because the automaton has to deal with more complex objects than just terrain, including many with vertical sides and overhangs. However, such objects can be handled to a certain extent by appropriate labelling of lines. There are also some problems of constant-elevation sensing (discussed below), but at the moment the contour-interval method appears to be a good basic structure for the metric modeling of objects.

One problem that must be overcome is the fact that ranging at a constant tilt (other than zero-tilt) is not the same as ranging at a constant elevation. Thus for panning when the ranging device is tilted, either the data points come back at different elevations, depending on the tilt, or else feedback control of a variable tilt angle must be employed. It might be that most ranging at variable elevations should be done at a constant bearing; in this case, it should be adequate to use interpolation between actual data points to obtain data at desired elevations. (Also, contours at "floor" and "ceiling" elevations could readily be obtained from the TV-camera inputs.)

In two dimensions, each entity in the model is a list of lines. Correspondingly, in three dimensions, we would have a list structure, possibly with much branching and interlinking for complex objects. For a simple example, consider Fig. 6.

The four parts of Fig. 6 indicate sets of lines derived from range data at the four elevations: 1, 2, 3, and 4 feet. A linking of lines into a list (as probably being part of the same entity) is implied at each elevation (two lists at the 3-foot line).

Other range data besides that at the indicated elevations might be temporarily acquired in order to aid the decision process on what linking to do between the lists. Suppose a scan along a vertical line linking Points P_1, P_2, and P_3 indicated vertical continuity from below P_1 to above P_3. For such a case we might specify programming so that List B is linked to List A. Similarly, we might end up with Lists C and D as both sublists under List B, implying that a single entity is being portrayed. Actually, the example illustrated in Fig. 6 represents two oscilloscopes on a desk, but to consider such a group as a single entity is not an unreasonable error for purely metric modeling.

As a reminder that the other part of the dualistic model should not be forgotten, the fact of emptiness at the 4-foot elevation should of course be stored in the record of void spaces.

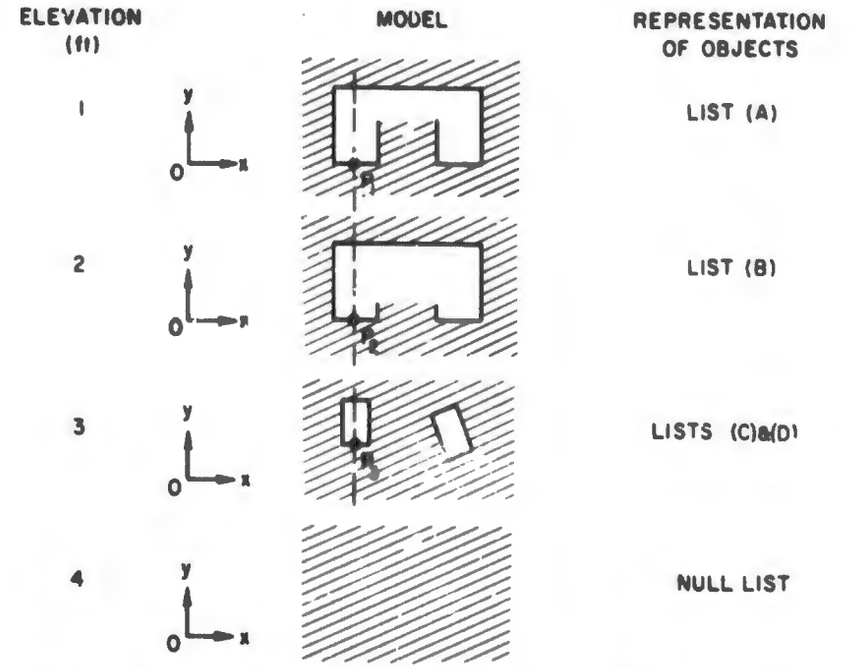


FIG. 6 EXAMPLE OF FITTING CELLS TO REPRESENT A VOID AREA

C. Further Work on Navigation Tactics

In the simulation work done for a continuous environment, the explicit aim has been to implement the GOTO mode. But implicitly, this work could almost as well be called the EXPLORE mode. The reason is that we have generally assumed no initial knowledge (or initial model), therefore requiring the automaton to "explore" in the process of "going to" the goal. Vice versa, if the primary aim had been exploration, it would have been necessary to program GOTO tasks in order to support EXPLORE routines. The objective would have been much the same--namely, the building of a model based on sensory data collected in the process of going from one place to another.

Although not always identified as such, the work done could be broken down into some on the strategic level, and a greater amount on the tactical level. On the strategic level the programming has involved simulation of movement to and around corners, when possible--or when not possible, consulting a tactic for a new move--based on a finite-width vehicle. On the tactical level, several methods have been employed for generating candidate points, and then selecting one of these for a move, as part of the GOTO task.

Initially, there was no model-building of any type, except for keeping a record of points visited. Each GOTO tactic was based only on this

"track" of past movements, along with the current two-dimensional view of the environment. Nevertheless, several flow diagrams for the GOTO task were generated on the assumption that a two-dimensional, polygonal model could be constructed that amounted essentially to the superposition of all views of the environment taken up to the present point.

These flow diagrams were developed to varying degrees of depth and complexity, but in general had the following features:

(1) If goal is in view, go to it.

(2) If not, consult model to see if left or right endpoints of obstructing object are known.

(3) If one or both endpoints are known, they are candidate points for a move if in view, or subgoals if not in view.

(4) If either endpoint is not known (and hence also not in view), then the end of the object obstructing the view toward the unknown endpoint becomes a candidate point.

(5) Given one or more candidate points for a first step, the model may be consulted for the image (imagined view) from each of those points for derivation of candidate points for a second step, and planning of this type may be done to any depth desired (and similarly plans for reaching subgoals).

(6) If one or more branches terminates on the goal before planning has reached maximum depth, select the shortest complete route for trying. Otherwise, select the incomplete route ending "closest" to the goal (modified by past travels).

(7) Take the first step along the selected route. If new data is obtained at the new location, update the model and then repeat the procedure, possibly revising or extending the plan.

For the initial round of simulations, no modeling was done, and hence no planning was possible. Based on Items 1 to 4 above, two candidate points for a first step were generated, and the preferred one selected as indicated by Item 6. This tactic was named CHOICE3.

Later on, when a two dimensional model-building routine was available, work began on the tactic CHOICE4 for planning to a depth of two steps. This routine has been largely programmed but not fully debugged. It is now believed that for further work, this routine should be modified to make use of the dualistic type of model recommended above.

Although a dualistic model including a separate record of observed open spaces had been considered before, a problem with the CHOICE4 routine re-emphasized the need for it. It was found impossible to derive from the existing, unitary model the information as to whether an apparent endpoint

of an object in the model was a real endpoint to consider stepping to, unless that endpoint were also in view. True, the modeling of objects could be modified to incorporate tagging of real corners. But what one really wishes to know is whether there is a void space to move into in the vicinity of an endpoint of interest. From that observation, it is an easy extension to realize that it is just as important in general to record observed voids as to record and model observed objects--hence the recommendation of a dualistic model.

Aside from aiding the identification of real corners, the use of a record of voids to supplement the record of objects should have additional effects on strategy and tactics for the GOTO and EXPLORE modes. It is premature to attempt to predict just how far this influence will extend. However, one likely effect in general is that tactics for GOTO will in the future more often incorporate the use of traversals along or through "passageways" (e.g., like the "centroid" tactic described in Sec. II-C-4-e of the First Interim Report), as opposed to the emphasis on going to the ends of objects that is implicit in the type of planning outlined in Items 1 to 7 above.

V STATUS OF SPECIAL EQUIPMENT

The special equipment being designed and constructed includes the vehicle itself, the TV preprocessor, the range finder, and the communications system between the SDS 940 and the vehicle. The basic design considerations of these items were discussed at length in the first interim report. This section will review the present status of this equipment and any design changes made since the first report.

A. The Vehicle

The mechanical construction of the vehicle is expected to be largely completed by May, with the exception of the arms. The logic on board the vehicle has been designed, and wiring and fabrication of the electronics have begun. The basic vehicle is still expected to be connected to the computer in early April 1967, for debugging tests.

In order to minimize the cost and number of logic boards, all action modules containing stepping motors have been made part of a multiplexing system. The distance registers associated with the modules circulate in a common shift register. Access to and modification of a given distance register takes place by stopping the shift in a time slot associated with that distance register. The exact function of the multiplexing system will be described in conjunction with a more detailed technical description of the vehicle planned for the next interim report.

To preserve vehicle battery power, a power supply system on the vehicle will segment power into several units that can be disconnected under computer control. Even so, a calculation of the typical power load reveals that the lead acid batteries would discharge in about two hours under typical load. Since such a rapid discharge is deemed inconvenient, a nickel cadmium battery will be specified.

As mentioned in the Introduction, we have temporarily suspended work on construction of arms for the vehicle. The arm design has progressed to a point where approximately six man-weeks of detailed design remain. The additional cost to complete design and construction of the arms is estimated to be approximately $6,500, including $500 for material not yet ordered. A detailed description of the present arm design is included as Appendix B of this report.

B. The TV Preprocessor

Due to a temporary shortage of experienced logic designers and to some unavoidable delays in delivery of parts and logic modules, the TV preprocessor is now not expected to be completed until about May 1, 1967.

A digitizer for the video signal from the TV camera has been constructed and checked out. This unit allows direct input of a TV frame into the computer and will later be used as an input to the TV preprocessor.

A programming specification for the TV preprocessor has also been developed. This specification provides the programmers with a detailed input-output description as received by the input-output channel and by the computer program.

Code assignments were made for the special, added program instructions to which the preprocessor will respond. Some instructions enable initializing parameters to be set into the unit. Other instructions activate various processing actions, executed off-line with respect to the computer and the input-output channels. And still other instructions permit the computer to monitor the status of the preprocessor.

Interrupts are also described. These interrupts indicate the end of major processing activities.

C. The Range Finder

The present status of the range finder unit is as follows:

(1) The mechanical and optical components required to produce a 2-inch diameter collimated light beam, the rotating mirror used to deflect the light beam, and the photocell receiving optics are completed and mounted on the yoke that houses the TV camera. This unit has been tested and is ready for final installation on the automaton.

(2) The electronics required to amplify the reference pulse and the detected range pulse and generate a multivibrator gate output signal, whose length is a function of range, has been bread-boarded and tested with the mechanical and optical unit. The design has been completed and is now being constructed in final form on circuit cards suitable for installation in the automaton electronic package.

(3) The motor that drives the rotating mirror is a 60-cycle, 115-volt AC synchronous motor. For operation from the battery supplies on the automaton, a DC-to-AC converter will be required. The final selection and testing of this unit has not yet been completed.

A preliminary test to determine the functional relationship between range and output pulse lengths was performed and this data was provided to the programmers working on the simulation and programming studies. The output pulse length was measured for 20 range increments within the minimum and maximum ranges of 0.5 feet and 28 feet. Range increments of 0.5 feet were used from 0.5 to 3 feet, 1.0 feet intervals from 3 to 10 feet, 2 feet intervals to 20 feet, and 4 feet intervals from there to 28 feet. The time in milliseconds ranged from 1.53 milliseconds at 0.5 feet to 5.18 milliseconds at 28 feet. There is an approximate ±.005 millisecond variation at all ranges noted during short-term experimental tests. The final determination of range sensitivity at various ranges and overall range accuracy will depend upon the performance of the DC-to-AC converter that drives the synchronous motor, the final electronic packaging, and--to some degree--the overall mechanical environment in which the range finder operates when the automaton is in full mechanical operation. It is anticipated, however, that the overall range accuracy will be better than ±0.2 feet, at short ranges of a few feet, and should be within ±1 to 2 feet at maximum range. Final confirmation of these values will be made when the completed unit is finally tested.

D. The Communications System

The logic design of four major communications system elements has now been completed. These elements are: (1) the Mobile Control Unit (MCU), (2) the Fixed Control Unit (FCU), (3) the Console for Manual Control (CMC), and (4) the Interface Unit (IFU). The system positions of these units are shown in Fig. 7.

The MCU is on board the vehicle, and serves as a signal and data distributor for the many vehicle modules. The MCU handles the flow of all vehicle data to and from the computer, all vehicle-generated interrupt signals, and all computer-generated special command signals such as emergency stop and reset. Looking toward the vehicle modules, the MCU can serve a maximum of 64 modules, transferring up to 18 bits per module. A maximum of 7 vehicle-generated interrupt signals may be accommodated at present. Looking toward the computer, the MCU deals with 8-bit characters at any rate not exceeding about 100,000 characters per second. A maximum of 4 types of computer-generated special command signals can be accommodated at present.

The FCU serves to make the vehicle appear as a peripheral device to the computer's input-output channel, the Direct Access Communication Channel (DACC). The FCU also provides interrupt signals to the computer's interrupt chassis. Error detection facilities in the FCU and the MCU check the validity of the data and of the format of the character stream. Provisions have also been made for the eventual use of parity checks.

The CMU consists of a panel of lights and switches backed by a limited amount of logic. Logic in the FCU permits either the DACC or the CMC to obtain access to the vehicle, on an essentially non-interfering basis. A human operating the CMC console can issue commands directly to the vehicle and can monitor, in detail, the state of the vehicle.

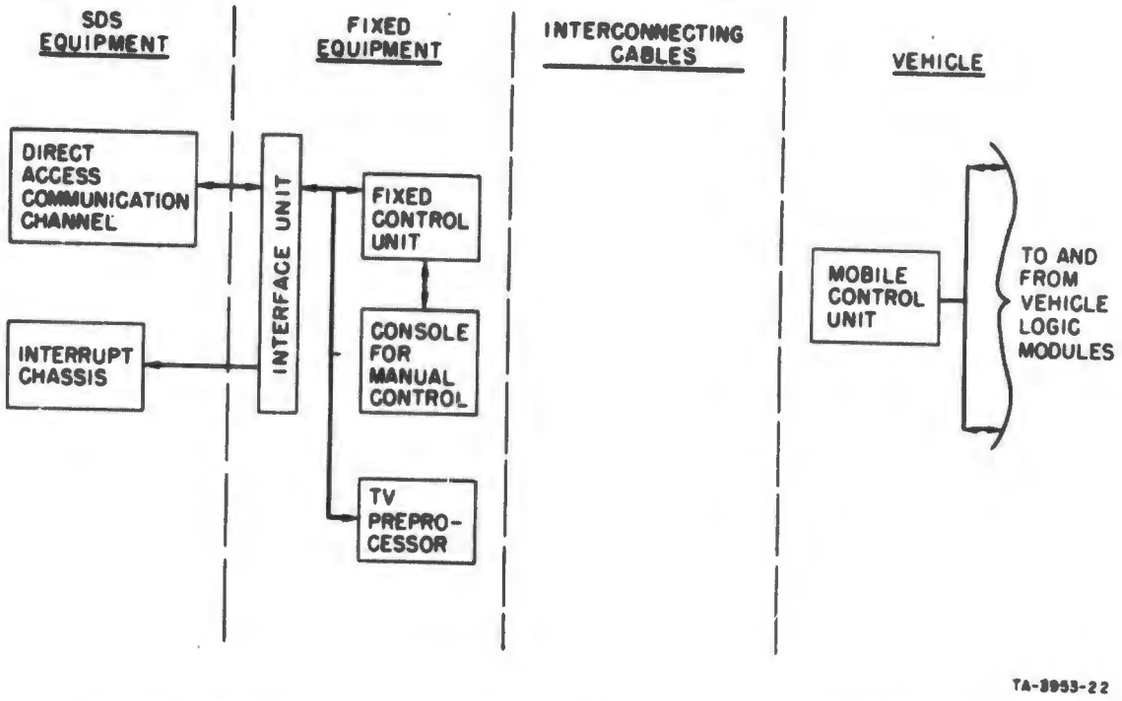


FIG. 7 POSITIONS OF MAJOR COMMUNICATIONS SYSTEMS ELEMENTS

The IFU provides a signal level conversion to allow interconnection of SDS circuits to the FCU and the TV preprocessor circuits. The IFU circuits also provide proper termination impedances to the SDS cables--they operate as transmission lines. The single IFU serves both FCU and the TV preprocessor, thereby providing economic use of cables and electronic equipment.

As an interim step, the FCU and MCU are to be interconnected by approximately 35 coaxial cables of very thin diameter, 0.1-inch O.D. (RG-174/U). These cables are supported at one end from the ceiling and are to be sufficiently long to enable vehicle movement over a rectangular area 37 by 28 feet located adjacent to the computer. By using the cable we expect to provide significantly easier system checkout and an earlier operational date than would be possible if radio links were used from the start. The design of the FCU and the MCU anticipates the use of radio circuits. Conversion to such circuits should require only the addition of equipment and almost no changes to the FCU, MCU, or computer programs.

The CMC also provides a means for debugging the system and programs. In conjunction with an on-line teleprinter the experimenter can use the computer to guide his actions on the CMC or to issue data via the DACC

and the FCU. Thus a very versatile operating procedure can be developed. Also, the CMC can control the vehicle even if the computer is unavailable or out of service.

A set of specifications has been drawn up describing the vehicle in terms useful to the programmer. Data formats, data codes, program controlled operations, and abnormal conditions and recovery procedures have been discussed in detail.

VI THE VISUAL SYSTEM

A. Introduction

It is anticipated that one of the major functions of the visual system will be to dichotomize the environment into filled or unfilled space. Filled space could be classified by object class for several classes of objects, especially those that have large planar surfaces and that are substantially differentiable from background and/or foreground clutter. Specific members of each class may be recognized by their location with respect to nearby objects. Near vertical edges of objects--e.g., door frames--or the presence of close objects can also be extracted for maneuvering aids. Specific landmarks may be recognized for navigational aids.

In order to select the correct tactics for executing commands, the problem-solving programs may need to know the type of space the automaton is in: open, or with sparsely distributed objects, or enclosed like a corridor or maze. It is expected that this categorization can be accomplished by using information from the visual system as well as from internally stored models of various levels of description of the environment. Tactile sensors and the range-finder scan also provide data for this task.

B. Status

Visual system efforts for the automaton can be placed in three overlapping categories: (1) system design, (2) hardware fabrication, and (3) automaton programming. The first two are currently underway and the third is beginning.

System design can itself be separated into two parts: (1) pattern recognition techniques, and (2) corresponding hardware design. The major hardware components have been obtained. The major items of the system are the TV camera and the preprocessor that extracts local features. The pattern recognition techniques can be further subdivided into roughly three highly overlapping categories: (1) preprocessing, (2) context constraints, and (3) decision making.

Preprocessing techniques have been specified to the extent that enables hardware to be built. (The reader is referred to the First Interim Report[1] for a complete description of the preprocessing scheme.) This specification has been accomplished with the aid of a considerable amount of computer simulation with real three-dimensional objects.

The techniques for using contextual constraints are currently under investigation. Four methods have been proposed, one of which a

complicated ad hoc scheme already shown by Roberts[2] to work for simple objects, has not been tried on our data; another, a clustering technique, has been tested and tentatively rejected; a third, a combinational technique, gave expected results for simple objects; and a fourth, a list-oriented technique, has also not yet been tried. It is expected that the third and fourth methods will be among the first to be converted into programs for the automaton.

Decision techniques differ widely in approach, from, for example, an active process of hypothesis-testing using attributes to passive trainable classifying networks. Some type of trainable network classifier, as well as a list structure technique based on topological properties, will probably be implemented in the first automaton system.

We have been using, for about a half-year, the TV camera that will be mounted on the vehicle. It has performed as well as a vidicon camera can be expected to. The drawbacks of this type of input device are several, but they are less restrictive than for other devices in terms of portability, availability, and cost. A fast A/D converter has been fabricated and will shortly be tested, connected directly to the SDS 940 computer. This facility will speed the rate at which visual data can be collected by more than an order of magnitude. This will allow the on-line development of specifications, using the SDS 940's scope display.

Writing automaton programs for operating visual hardware from the SDS 940 has commenced. The so-called "driver programs" that interface directly with external hardware are being written in assembly language. Subsequent SDS 940 programs will be written as their specifications are developed.

The remainder of this section discusses (1) changes in the preprocessing scheme from those reported earlier, (2) contextual constraint schemes, and (3) some suggestions for the functions of the first programs for the automaton's visual system.

C. Changes to the Visual Preprocessor

There are five changes in the visual preprocessor which resulted in simplifications in the hardware and processing procedures. Simulation results have shown that these changes will not significantly deteriorate the preprocessor performance.

First, the four samples used to compute a picture element are now spaced at alternate sample times on alternate scan lines for the full field format. Figure 24 of the First Interim Report showed consecutive samples on consecutive scan lines used; in this mode it would have been possible to miss a sharp edge between groupings of four samples.

Second, the permanently wired-in feature of "all-ones" will be replaced by using an all-ones feature as the first feature in each programmed feature set. This reduces the number of D/A converters required in the correlator.

Third, the first feature determines one of two thresholds, the other being specified by the computer. The difference between the highest correlation value and the average correlation value (over the feature set) is compared with the larger of the two thresholds. If the difference exceeds this threshold it is converted into digital form. Otherwise the feature response word is set to zero. Previously, two thresholds were used: (1) to test absolute value of the highest correlation, and (2) to test the ratio of the highest to the average.

Fourth, the facility to specify a test for zero at each picture element corresponding to the center of the feature will be included. When differentiated pictures are used, it is possible to save time by skipping the correlations for the zero case.

Fifth, only that feature having maximum correlation over the nine translates is recorded. Formerly, the maximum and second maximum were recorded. A second pass through the data can now be made, however, inhibiting a response at that translate at which a maximum previously occurred, thereby obtaining a secondary maximum, but at a different neighboring translate. It is expected that this secondary feature may help to resolve ambiguities when applying contextual constraints on the primary feature responses and provide higher resolution where necessary. In connection with this change, the format of the feature response word in the external core memory has also been changed.

D. Methods to Apply Context

The work "context" is used herein to denote information relating to probable types of associations between neighboring feature responses. This part of the visual processing is especially sensitive to the nature of the environment. The initial visual world of the automaton will be within an office building having mostly man-made objects. A common property of such construction is rectangularity. For example, most floors are flat, and walls are vertical planar surfaces. Man-made objects are put together from basic parts, which can often be described parametrically through functional expressions, rather than requiring a point-by-point description. The simplest function is the line that can be defined by four parameters on a two-dimensional projection. The few explicit methods now developed make heavy use of line synthesis techniques. Four methods of synthesizing longer lines from short line segments are briefly described in the remainder of this section.

A technique of combining short line segments into longer lines and then synthesizing line drawings of complete objects has been developed by L. Roberts.[2] This method consists of a multi-stage application of ad hoc techniques that appear to work for simple figures. Pattern recognition in this method is more a by-product than the main purpose, which is to create from a single gray-scale projection of a 3-D object--e.g., a photograph--a line drawing suitable for CRT display. This drawing can then be manipulated through various geometric transformations. One of the important aspects of this method is the use of a system of four-dimensional

matrices which specify the transformations. These matrices are sparse, which enhances their use in actual computation. It can be argued that Roberts' method does more than what is strictly necessary for pattern recognition. If the purpose is only to recognize, a complete line drawing may not be necessary. The location and orientation of the object may be a required output of the visual system for use in the internal model of the environment. If so, a completed description is required by Roberts' method, because a match is required over the complete visible portion of the object. It is now felt that object classification plus an approximate location of, and gross description of, the extent of the space occupied by an object will suffice. Details for each object can be obtained by additional sensing when required, and when close to the object. Thus Roberts' method, while interesting, is not an exact match to our requirements and will not be applied in its complete form.

The second method for applying context uses a clustering technique. Each feature response is a line segment and is described by two parameters. These parameters are calculated by first passing a line through and in the direction of the line segment. One parameter is the angle, θ, of this line with respect to a horizontal axis; the other parameter is the perpendicular distance, p, from the line to an origin. Other responding line segments on the same line will have the same values of p and θ. Minor variations in position and angle would produce a cluster of points. The sample mean of each cluster could be used to generate a line fitting many responses that are essentially collinear. Clusters having a small number of sample points would be eliminated as noise. These lines would be terminated at intersections with other such lines by testing for the occurrences of responses on each line segment between intersections. Experiments have shown that this method suffers from the quantization of the direction of the line-segment features. Usually the features have only eight or possibly twelve discrete directions. Clustering algorithms are more appropriate when distances can be measured on continuous variables. Also, a single contour falling between two directions may produce two types of feature responses, therefore producing more than one line by the clustering technique. Furthermore, even if minor variations in direction could be obtained, they would produce large variations in perpendicular distance when their location is not near the perpendicular line. For these reasons, this method will not be investigated further for use with the automaton's visual system.

The third method is patterned after biological examples.[3] Larger features are extracted by combining the smaller features in two steps. A large feature may be, for example, a vertical line extending over many subfields. Such a large feature would be synthesized if, in the area which it covered, there was a sufficient proportion of subfields having small feature responses that were either vertical or near-vertical and nearly collinear. These responses could be either logically combined using binary relations or linearly combined using weighting factors which weight collinear, vertical features more than other small features which are nearly collinear or vertical. It also is possible to extend these lines until this proportion factor falls below a threshold value, thus finding their extent. The longer lines would be fewer in number and have

the same possible directions as for the small line features. Initial experimentation has suggested that this method may be useful for part of the requirements of the automaton's visual system and is therefore being studied further.

The fourth method is similar to contour-following algorithms. Usually contour followers work on unprocessed visual data, which can require lengthy computation. This problem is minimized in the automaton's visual system because the original visual samples are transformed from a 240 x 240 picture space to a 38 x 38 feature space. Furthermore, the most likely direction of the contour is provided at each responding point by the feature extraction of the hardware preprocessor. The basic contour follower employs a small window which moves in a direction specified by the data in the window. As soon as a new response is found that can be accepted for extending the contour, it must be checked for previous occurrence in order not to be trapped in loops. Possible branch points should be noted so that search for additional contours can be more efficient. To find vertical lines in this manner should be fast because of the ordering of the response data by the preprocessor. This method will be investigated for possible use in complementing the third method described above.

E. Anticipated Functions of the Automaton Aided by Information from the Visual System

It is proposed that the visual system may be of assistance in the following functions. The remainder of this section contains comments regarding the manner in which this may be done.

(1) Classify environment into type--e.g., sparse, cluttered, etc.

(2) Dichotomize environment volume into filled or unfilled space.

(3) Classify objects in filled space having large planar surfaces, differentiable and isolated from other objects, background, and foreground clutter.

(4) Identify specific member of class by spatial ordering.

(5) Locate near-vertical edges of objects or closures.

(6) Notice the presence and approximate location of close objects.

1. Classification of Environment

Assume three types of environment: (a) sparse, (b) cluttered, or (c) a passageway. The initial environment may consist of a ground plane (floor), limiting boundaries (walls), filled spaces (objects), and unfilled spaces (air). Procedure for classification of the environment is to scan first with the range finder at zero tilt (level) or with the TV camera along the intersection between the ceiling and the walls, to

locate distances to walls. If the distance between any opposite pair of walls is small, then a class (c) environment exists. If not class (c), then scan the ground plane with the TV camera to obtain frequency of occurrence and approximate locations of objects on floor (no objects floating or with much overhang). If percentage of unfilled floor space is high, then environment is class (a); otherwise, class (b). It may be more reliable to classify on the basis of views from several locations.

2. Dichotomization of Space into Filled and Unfilled Regions

It may be possible to lower the tilt of the range finder and scan to provide a distance plot to the reflecting surface along what would be a sector of the surface of a cone, and then label relatively close sections as "filled" and far sections as "unfilled." It would be more definitive to obtain the approximate shape of the filled section by defining its near-vertical and top and bottom boundaries. To track these boundaries by using the range finder and the pan and tilt mechanisms would be less accurate and more time consuming compared to tracking them by program (if visible) in the stored visual field of the camera. The camera inputs a 24 x 24 degree view in 33 ms, and produces feature extraction in about 100 ms. These features could be combined, searching for grosser boundaries, on the order of an estimated few seconds. Because a 1:1 mapping exists between points on the floor and the stored projection, approximate location (with respect to the pan, tilt, and vehicle orientations) of the objects resting on the floor could be obtained. By geometric means a planar map could be produced locating (approximately) filled vs. unfilled space on the floor.

3. Classifying Objects

Two techniques appear feasible for classification of objects on the basis of their visual properties. Both are based on shape features obtained from contours or intensity gradients from a single black and white view, although intensity of large surfaces may be included at a later date. If funds and time permit, stereo may also be tried, but only after a complete monocular vision system exists.

An attempt at closure around the closest object (as determined by position on the floor) will be made first. The closure routine will not require 100% continuity, which in general is impossible to provide. This routine will be similar (possibly the same) as the routine for (2) used to delineate filled space; if (2) precedes (3), closure of the closest object will already have been defined. All responses outside the closure will be ignored.

One classification technique will use trainable networks that will make decisions based on training with a large data base. The classification technique should also include a "don't recognize" response. Distance in pattern space is the important metric in these circumstances, and it is expected that features can be chosen such that distance will be a reasonable measure of similarity. However, it is probable that some views of objects in one class are "closer" in pattern space to certain

views of objects in other classes than to other views in the same class. It is imperative, therefore, if such a scheme is to be used, that classes be divided into subclasses corresponding to the several shapes found in each class. Clustering techniques may be adequate for this task.

A second technique is also a candidate for programming for the first system. It describes an object class on topological properties--e.g., a cube has a Y-type vertex closest, below the camera, branching to three other Y-type vertexes connected through L-type vertexes along the boundaries. Again, many such descriptions of a cube will exist, depending on the viewing angle and position. This technique may use different preprocessing and context application methods. Because vertexes are important, it may be necessary to inquire in the original picture data as to the type of vertex, using different feature sets than were used to generate the line segments. It must be emphasized that state-of-the art pattern recognition will only be useful in a restricted environment. Specifically, it may be possible to obtain reliable decisions only if the objects are simple (no smaller surfaces than approximately 2^0 of arc of a total of 24^0), nonoverlapping, planar-surfaced, and distinct from background. Existing pattern recognition techniques for handling overlapping objects normally assume complete and error-free line drawings as inputs, the presence of which cannot yet be guaranteed.

4. Identify Specific Member of Class

This topic is discussed only to mention that the use of subclasses to identify specific members of classes by their visual properties alone may not be as easy as the use of a higher level program that would use location of the object. The visual system may be used to check the model, however, by noting the relative ordering of objects from a given location.

5. Locate Near-Vertical Edges

Because the hardware preprocessor orders the data in columns, connecting local features in columns should be faster than other combinatorial operations. Hence near-vertical edges could be the first output from the visual system. These edges can be accurate to 1/10 degree as opposed to 1.8 degree for the range finder. This may be useful for navigating through doorways or between close objects.

6. Notice the Presence of Close Objects

The technique for noticing close objects would be to look for long lines close to the bottom of the field of view. Knowing the camera height, tilt angle, and lense acceptance angle the distance of any point on the floor may be computed. Again this procedure is elementary compared to (3) and should not take lengthy processing. It may be useful for coming close to an object without actually touching it. The range finder could be used to spot check this output of the visual system.

Appendix A

DISPLAY PACKAGE FOR SDS 940 SCOPE: TENTATIVE SPECIFICATIONS

Appendix A

DISPLAY PACKAGE FOR SDS 940 SCOPE: TENTATIVE SPECIFICATIONS

by
J. R. Bell

Our aim is to propose a set of display programs which facilitates sophisticated use of the SDS-940 scope. At the same time compatibility has been maximized among this set, the existing XTRECØPE package on the CDC 3100, and the suggestions sent to us by Bolt Beranek and Newman (particularly the idea of labels).

The data organization chosen is a form of tree structure (of buffers and pointers) as will be detailed in a moment. This appendix consists of three sections: this section explains the structure of the "descriptor trees," the next section presents the routines for manipulating the trees, and the final section shows how to create the display buffers which comprise the lowest levels of the trees.

A. Structure of Descriptor Trees

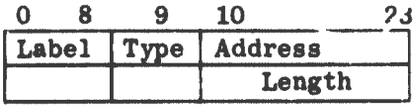
1. Definition of Terms

(1) "Buffer" (or "display buffer") is the block of core containing the bit pattern sent to the display. The format of the buffer words varies with the scope used. For the SDS-940 scope, see the SDS manual 900987A.

(2) "Descriptor" (or "display descriptor") is a pair of memory words containing a pointer (i.e., address) together with some subsidiary information about the size and nature of what it points to. More details follow.

(3) "Descriptor list" is a sequential set of descriptors.

2. More About Descriptors



A descriptor has four subparts, located as shown in the above diagram. The address and length fields specify the location and size of the entity pointed at by the descriptor. This entity can be either a buffer or a descriptor list. These cases are represented by zero or one, respectively, in the type field.

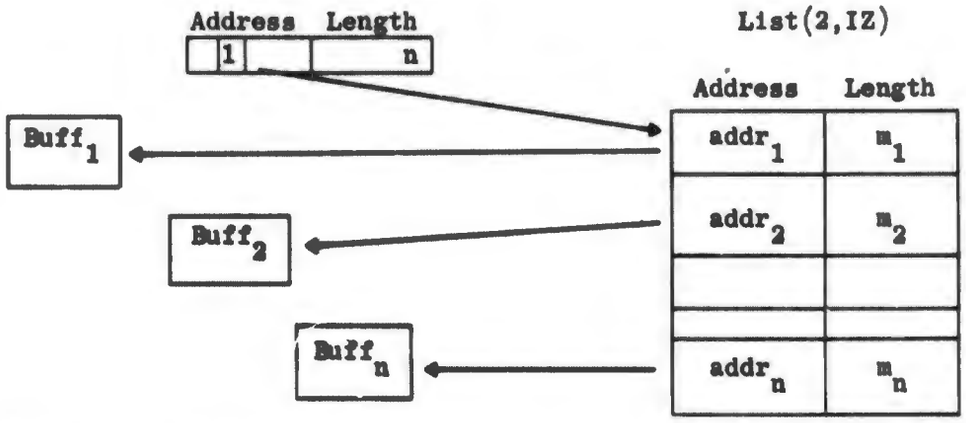
The label field contains a nine bit integer. This integer is a subscript used to index a label table. The label table contains alphanumeric identifiers. Conceptually, one can visualize the identifier itself as appearing in the label field of the descriptor.

By convention, the label index zero corresponds to not being labelled.

The labels facilitate handling a hierarchy of user-defined entities and subentities in lieu of the primitives (point, line, and character). A label may contain from one to eight alphanumeric characters. They are normally Hollerith constants of the form 5HSAMPLE. Such constants can also be used as values of real variables. Alphanumeric I/O is possible using the A format.

3. Examples

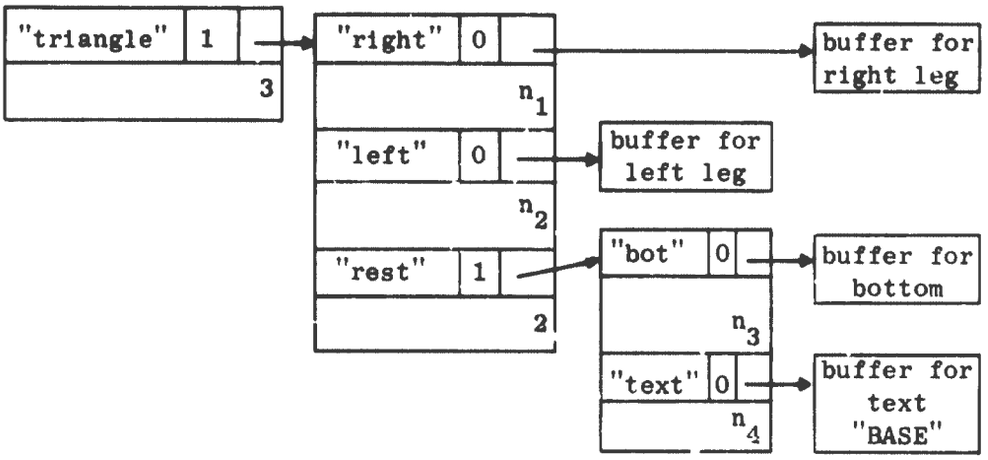
(1) A list such as in XTRECØPE



As is shown in the diagram, the conventions have been chosen to maximize compatibility.

(2) A labeled list (a la BBN) to form a triangle with sides named right, left, and bottom and the bottom labeled BASE.

We might have



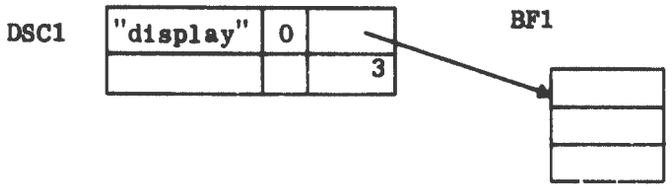
In essence, we have created a tree structure. We can now manipulate any substructure, using the name of its principal node as a handle.

B. Display Service Routines--Descriptor Handling

(1) DESCTR

DESCTR (Variable, label, pointee, size of pointee, type of pointee) Initializes the subfields of a descriptor. Variable is the name of a two-word array; the other four arguments correspond to the four subfields of a descriptor. There are also separate routines handling the individual subfields.

Example: CALL DESCTR (DSC1, 7HDISPLAY,BF1,3,0)



The actual parameters for variable and pointee may be elements of an array. Thus, for example, if LIST was a 10 X 2 array representing a descriptor list then

CALL DESCTR(LIST(7,1),1HL,B,5,0)

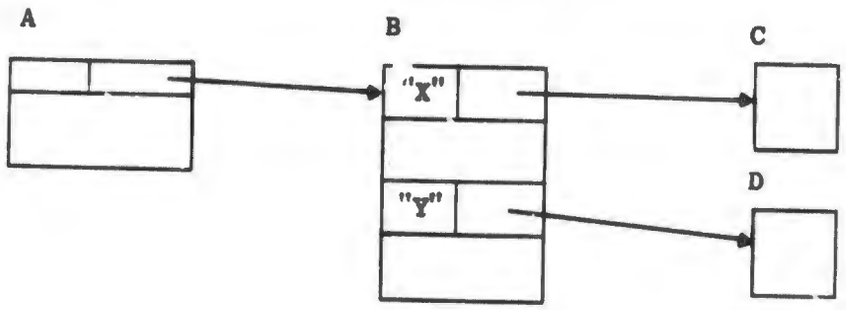
is a way to set the subfields of the seventh descriptor in the list.

(2) SETSIZ, SETTYP, SETLBL, SETPNT, CHKSIZ, CHKTYP, CHKLBL, CHKPNT

The subroutine SETSIZ (variable, size of pointee) or the function CHKSIZ (variable, size of pointee) sets or returns the "size" subfield of a descriptor. Analogously there are SETTYP, SETLBL, SETPNT, CHKTYPE, CHKLBL, and CHKPNT for the other subfields.

(3) DENY

By using SETSIZ (DESCRIPTOR, 0) a descriptor pointer's power to access is temporarily erased. The subroutine DENY (variable) is equivalent to this. For example:



DENY (B(2,1)) will erase the access to D granted by the descriptor in list B. The access could later be restored using a SETSIZ.

(4) LABLØC

The function LABLØC (variable, label) returns the location of the descriptor with a given "label" occurring in the structure or substructure headed by "variable."

Example: In the last diagram DENY (LABLØC (A,1HY)) would be equivalent to

DENY (B(2,1)).

Using the SET and CHK operations together with LABLØC any desired manipulation of the descriptor tree can be accomplished.

(5) DISPLAY (variable)

The subroutine DISPLAY (variable) displays all the buffers accessible through the tree headed by the descriptor variable given as the parameter.

Example: A program to display a buffer B of length n

```
CALL DESCTR(D,B,n,0)

CALL DISPLAY(D)
```

Example: To display the subpart labeled "Y" in the last diagram we could have

```
CALL DISPLAY (LABLØC(A,1HY))
```

(6) PREDISPLAY

Prepares monitor to drive the display.

(7) UNDISPLAY

Stops the display. Unfreezes memory. Ignored if display is not on.

(8) LIGHTPEN (variable)

This is a function whose value is the label (i.e., string name) of the part of the display which caused the lightpen interrupt. The parameter is a descriptor variable as in DISPLAY.

(9) CØØRD (X,Y)

Enables the raster (which is normally disabled), finds the coordinates of the lightpen, returns them via x and y, and then disables the raster.

C. Buffer Service Routines

These routines create buffer words for the display, append them to the indicated buffer, and update the length field of the descriptor pointing at the buffer.

(1) BUFFER (descriptor)

This routine specifies the buffer being referred to in subsequent steps. It remains in force until changed explicitly.

(2) PØINT (x,y)

Displays a point at position x,y. The parameters must have integer values between 0 and 1023. (One buffer word is created.)

(3) IPØINT (x,y)

Creates an intensified point at x,y. Otherwise identical to PØINT.

(4) BEAM (x,y)

Position beam at x,y but do not display. Creates one buffer word.

(5) VECT (x,y)

Draw a vector from present position to x,y. One buffer word.

(6) IVECT (x,y)

Same as VECT except that vector is intensified.

(7) CHARSIZE (n)

Determines the size of future characters until explicitly changed. The parameter must be 1,2, or 4. These represent normal, double, and quadruple size, respectively. (One buffer word.)

(8) CHAR (1HA)

This generates any symbol "a" on the scope. The size of the character is the last CHARSIZE. The position of the first of a series of characters is specified using BEAM. Succeeding ones appear left to right, typewriter style. One buffer word.

(9) STRING (x,y,n,loc)

Takes n characters packed 4 per word starting at loc and displays them starting at x,y. One buffer word per character.

D. Sample Program

This sample program displays on the scope a polygon with a comment underneath. The number and location of the vertices, and the comment, are input from the keyboard at run time.

```
      ARRAY K(2),L(40), IX(20), IY(20)
      ACCEPT 10,N
10    FORMAT (I2)
      DØ 20 I=1,N
      ACCEPT 15, IX(I), IY(I)
15    FØRMAT (2I4)
20    CØNTINUE
      CALL DESCTR (K,7HEXAMPLE,L,0,0)
      CALL BUFFER (K)
      DØ 30 I=1,N
30    CALL VECT(IX(I),IY(I))
      CALL PREDISPLAY()
      CALL CHARSIZE(4)
      CALL BEAM(100,100)
```

```
40    CALL DISPLAY (K)
      ACCEPT 50, ICHAR
50    FORMAT (A1)
      CALL CHAR(ICHAR)
      IF (ICHAR-1H.) 40,60,40
60    CALL UNDISPLAY
      STØP
      END
```

A user might now insert from the keyboard the parameters

```
4
400,400
400,800
800,400
800,800
THIS IS A SQUARE
```

and get the picture

THIS IS A SQUARE

By adding a period to the end of his message he would signal to terminate his display.

Appendix B

MANIPULATOR ARMS FOR THE AUTOMATON

Appendix B

MANIPULATOR ARMS FOR THE AUTOMATON

by

V. Lieskovsky

The purpose of the arms for the automaton is to enable the vehicle to execute tasks involving a limited amount of moving, carrying, and lifting. (The arms might be similar to those of a crab.) We have found it helpful to refer to the different parts of the manipulator arms in anthropomorphic terms. Such reference does not imply that we have attempted to duplicate the corresponding functions of a human arm, but merely helps to clarify concepts by association. (Actually our use of anthropomorphic nomenclature is probably responsible for our thinking of exactly two arms for the manipulator instead of considering the possibility of using perhaps three or more.)

For simplicity we decided on two arms, hinged at the foremost and widest point of the vehicle. The arms consist of upper and lower arms, hinged at the shoulder, elbow and wrist. All these hinges permit rotation only in one plane. The shoulder has freedom to turn in the horizontal plane, the elbow in the vertical plane, and the wrist in a plane perpendicular to that of the elbow. The wrist joint actuates the manipulator proper, which is a rotable suction cup at the present, but may be replaced by a pair of clawlike pinchers if so desired.

The overall design philosophy has been to provide a pair of extremities, flexible in their operation. This flexibility is achieved in the following way. The arms fully extended, with a reach of three feet, may be initially kept fixed about their joints. With these stiff arms, the most basic functions of pushing objects around in the area by moving the vehicle accordingly may be accomplished under control of the computer. Once this function is under control, the next step may be rotating of the stiff arms around their shoulders only, with the vehicle at a standstill. Combined vehicle motion and shoulder joint rotation will bring about a more articulated means of retrieving an object by sliding it across the floor into a desired location. Progress in software development will then allow the use of the elbow (to lift the object) and the use of the wrist (to orient the manipulator proper to the most suitable position relative to the object); it will also permit the use of the manipulator to rotate the object) in order to inspect it visually. This design approach will save hardware costs in the long range, since only one arm design of ultimate forecasted complexity is needed. Another approach would have necessitated several arms of successively increasing complexity.

Mechanical efficiency requires that the smallest possible amount of mass is placed on the arm itself. This requirement would place the drive motors on the carriage, with some mechanical method of transmitting power to the joints. Looking into different schemes convinced us that the efficiency of practically any such mechanical system is so low that it would have drained the battery power beyond that allowed for this purpose. Therefore, the drive motors of the individual joints were located close to the joints but in such a fashion as to reduce inertial effects.

The drive motors themselves are of the stepping motor type. This decision allowed the use of direct digital control, and proved to be superior from the point of view of price and delivery. The digital control utilizes a pulse counting scheme for position determination. It now follows that a reference position is required at every actuated joint. These are limit switches in some cases and light-photocell pairs in others. Closed loop servo control of the motors is not planned because of the added cost and complexity. The advantage of such control would be an increase in operational speed. If reconsideration requires that alternative, this feature may be added without re-design of the mechanisms.

At every joint, a torque limiting scheme is utilized to protect the gear and the arms from damage. This function is accomplished by means of slip-clutches. Indication of slip condition is provided in order to inhibit action. It is a simple matter to equip the arms with strain gages to monitor force levels in the arms in order to protect the object to be handled from damage. Similarly, touch indication may be derived from these strain gages for the vehicle's safety or for other information.

In the locomotion state, the arms will usually be folded back alongside the vehicle and will extend only slightly behind the vehicle. This is the state that will probably be used when going through a doorway, etc.

Figure 8 illustrates the layout of the mirror-symmetric arms. Table I contains operational information about the arms.

Table I

OPERATIONAL CHARACTERISTICS OF THE MANIPULATOR ARMS

Item	Units	Shoulder	Elbow	Wrist	Cup
Transmission ratio		25:1	50:1	12:1	25:1
Steps/rev of motor		200	200	200 or 100	24
Joint motion	•/step	0.072	0.036	0.150	0.60
Full stroke	•	215	130	210	Continuous
Step rate	1/sec	100	100	100	100
Steps/• of joint		13.89	27.78	6.67	1.67
Rate at joint	•/sec	7.2	3.6	15.0	1.83"/sec*
Time of full stroke	sec	29.86	36.11	14.0	Continuous
Torque at joint	ft-lb	10.74	7.81	0.985	0.146
Force at end of member	lbs	3.58	6.70	3.94	1.0

*Peripheral speed of cup.

REFERENCES

1. C. A. Rosen and N. J. Nilsson (Eds.), "Application of Intelligent Automata to Reconnaissance," First Interim Report, Contract AF 30(602)-4147, Stanford Research Institute (November 1966).

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

3. D. H. Hubel and T. N. Wiesel, "Receptive Fields, Binocular Interaction, and Functional Architecture in the Cat's Visual Cortex," Journal of Physiology, Vol. 160, pp. 106-123 (1962). Also in Pattern Recognition, L. Uhr (Ed.) (John Wiley & Sons, New York, 1966).

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 Inst.
Menlo Park, Calif.

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE
Application of Intelligent Automata to Reconnaissance

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Interim Report November 66 - March 67

5. AUTHOR(S) (First name, middle initial, last name)
Charles A. Rosen Nils J. Nilsson

6. REPORT DATE
September 1967

7a. TOTAL NO OF PAGES
70

7b. NO OF REFS
3

8a. CONTRACT OR GRANT NO
AF30(602)-4147

b. PROJECT NO
4594

c. Task No.
d. 05

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
RADC-TR-67-171

10. DISTRIBUTION STATEMENT
This document is subject to special export controls and each transmittal to foreign governments, foreign nationals or representatives thereto may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440

11. SUPPLEMENTARY NOTES

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 four 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 automation 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.

As of the beginning of March 1967, it is anticipated that the vehicle and other special hardware will be completed and be ready for preliminary experiments within two months. The preprocessor is expected to be completed a few weeks thereafter. Various "Immediate Action" computer programs are now being written to drive the registers on the vehicle and thus accomplish simple movements and sensing acts. In addition, maneuvering tactics designed during previous simulation experiments are being written into SDS 940 computer programs that will call the immediate action routines in the proper sequence to accomplish simple missions.

DD FORM 1 NOV 65 1473

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Pattern Recognition						
Reconnaissance						
Automata						
Decision Theory						