SRI International

SHAKEY THE ROBOT

Technical Note 323

April 1984

Edited by: Nils J. Nilsson, Director

Artificial Intelligence Center
Computer Science and Technology Division

Approved:

Donald L. Nielson, Acting Director
Computer Science and Technology Division

CERTUM QUOD FACTUM

Giambattista Vico — Italian philosopher and jurist (1668-1744)

CONTENTS

LIST OF ILLUSTRATIONS v

LIST OF TABLES vii

ABSTRACT 1

CHAPTER ONE: *Introduction* 3

CHAPTER TWO: *The Robot Vehicle, The Computers, and Other Hardware* 9

CHAPTER THREE: *Shakey's Model of the World* 19

CHAPTER FOUR: *The Low-Level Actions* 25

CHAPTER FIVE: *The Intermediate-Level Actions* 35

CHAPTER SIX: *Vision Routines* 51

CHAPTER SEVEN: *STRIPS* 57

CHAPTER EIGHT: *Learning and Executing Plans* 65

CHAPTER NINE: *Experiments With Shakey* 81

ACKNOWLEDGMENTS 101

APPENDICES

A *Mechnical Development of the Automaton Vehicle* 105

B *Some Current Techniques For Scene Analysis* 113

REFERENCES 135

ILLUSTRATIONS

1	AUTOMATON VEHICLE	10
2	AUTOMATON VEHICLE IN ITS ENVIRONMENT	11
3	AUTOMATON-SYSTEM BLOCK DIAGRAM	14
4	SRI ARTIFICIAL INTELLIGENCE GROUP COMPUTER SYSTEM	17
5	EXAMPLE MODEL	24
6	CONTROL STRUCTURE OF LOW-LEVEL ACTIVITIES	33
7	CONTROL STRUCTURE OF THE INTERMEDIATE LEVEL	48
8	AN OBSTACLE CONFIGURATION FOR FINDPATH	50
9	SEARCH TREE FOR CONFIGURATION OF FIGURE 8	50
10	TYPICAL MACROP	67
11	MACROP WITH MARKED CLAUSES	70
12	MACROP AFTER EDITING	71
13	GENERALIZED PLAN FOR TWO-PUSH MACROP	74
14	MOP: A NESTED MACROP	77
15	MAP OF SHAKEY'S EXPERIMENTAL ENVIRONMENT	83
16	MAP OF SHAKEY'S WORLD AFTER COMPLETION OF THE FIRST TASK	96
B-1	THREE CORRIDOR SCENES	116
B-2	RESULTS OF MERGING HEURISTICS	117
B-3	A SIMPLE SCENE	124
B-4	A MORE COMPLICATED SCENE	127
B-5	THE ANALYSIS TREE	129

B-6 BASIC FLOWCHART FOR LANDMARK PROGRAM 132
B-7 LANDMARKS 133

TABLES

1	PRIMITIVE PREDICATES FOR THE ROBOT'S WORLD MODEL	23
2	LOW-LEVEL ACTIVITIES OF ROBOT*	32
3	SUBROUTINE GOTOADJROOM (ROOM1,DOOR,ROOM2)	37
4	INTERMEDIATE LEVEL ACTIONS	40
5	MARKOV TABLE FOR THE LOWEST-LEVEL PUSHING ILA	47
6	AXIOM MODEL	84
7	STRIPS OPERATORS	89
8	TRIANGLE TABLE FOR MACROP1(PAR3,PAR1,PAR2,PAR4,PAR5,PAR7,PAR6)	93
9	TRIANGLE TABLE FOR MACROP2(PAR3,PAR1,PAR6,PAR7,PAR5,PAR4,PAR2)	98
B-1	CORRESPONDENCE BETWEEN BOUNDARY SEGMENT CONFIGURATIONS AND CHARACTERS USED IN PRINTOUT	118
B-2	REGIONS THAT ARE LEGAL NEIGHBORS	120
B-3	HYPOTHETICAL REGION SCORES	128

ABSTRACT

From 1966 through 1972, the Artificial Intelligence Center at SRI conducted research on a mobile robot system nicknamed "Shakey." Endowed with a limited ability to perceive and model its environment, Skakey could perform tasks that required planning, route-finding, and the rearranging of simple objects. Although the Shakey project led to numerous advances in AI techniques, many of which were reported in the literature, much specific information that might be useful in current robotics research appears only in a series of relatively inaccessible SRI technical reports. Our purpose here, consequently, is to make this material more readily available by extracting and reprinting those sections of the reports that seem particularly interesting, relevant and important.

CHAPTER ONE

Introduction

From 1966 through 1972, the Artificial Intelligence Center at SRI conducted research on a mobile robot system nicknamed "Shakey." This research was sponsored by the Advanced Research Projects Agency under a succession of contracts with the Rome Air Development Center, the National Aeronautics and Space Administration, and the Army Research Office. Two complete versions of Shakey were developed. In 1969 we completed our first integrated robot system: a mobile vehicle equipped with a TV camera and other sensors—all radio-controlled by an SDS-940 computer. In 1971 we completed a more powerful robot system by making substantial program improvements and by replacing the SDS-940 computer with a Digital Equipment Corporation PDP-10/PDP-15 facility.

Dramatic recent progress in reducing the size and cost of powerful computer hardware makes the prospect of autonomous robots much more realistic than it was fifteen years ago. There are several new robot projects underway that might benefit from Shakey's legacy. The Shakey project led to several advances in AI techniques, many of which were reported in the literature, but a great deal of specific information nevertheless appears only in a series of relatively inaccessible SRI technical reports [1-12]. Therefore, to make this material more readily available, we have decided to extract and reprint here what seem to be the most relevant and important sections of these reports. Of particular interest are (1) the techniques used in Shakey's action routines that enabled flexible recovery from inappropriately executed actions, (2) the method of integrating perception with action, and (3) the techniques for planning and executing complex sequences of actions. (The reader who needs additional details can obtain copies of the original reports from the National Technical Information Service (NTIS). See the NTIS access numbers in the references at the end of this report.)

This report will describe only the second of the two Shakey systems because it was far more advanced than its predecessor. (A summary of the first system appears in [5].) The material is reprinted in its original form, but with minor changes to make figure, chapter, and citation numbers consistent. Whenever deemed advisable and helpful, the text is supplemented by occasional explanatory comments in italics. Unless otherwise attributed, any chapter or section references included in these commentaries pertain to the present collection only.

We begin with an excerpt from the first report [1], issued in 1966.

Major Goals and Objectives of this Program

It is the objective of this program to develop concepts and techniques in artificial intelligence enabling an automaton to function independently in realistic environments. These concepts shall be demonstrated by means of a breadboard, mobile vehicle containing visual, tactile, and acoustic sensors, signal processing and pattern-recognition equipment, and computer programming. Primary goals shall be the solution of incompletely specified problems (requiring creation of intermediate strategies and goals) and improvement of performance with training experience.

Some of the ground rules guiding our research were established immediately. First, it was decided that the basic goal of this project was to design an integrated system consisting of a mobile vehicle under the real-time control and supervision of a powerful digital computer. The vehicle should be equipped with at least rudimentary manipulative abilities, and with sensory and communication subsystems. Various automata have been built which are controlled by relatively few, simple, onboard logic circuits, but the essence of this project is real-time control by a full-scale, programmable, digital computer.

Second, we decided to minimize hardware complexities whenever possible to allow us to focus primary attention on the problem of directing the automaton's actions and planning by means of a hierarchy of computer programs. For this project the mechanical engineering problems of building a robot with articulated limbs and delicate grasping abilities are irrelevant. One can face very tough problems in artificial intelligence directly in attempting to write computer programs to control even a very simple vehicle. It is for

this reason also that we shall make no attempts here to design highly miniaturized computers that can fit into the "head" of an automaton. Technology will sooner or later provide us with such small but powerful computers in any case; in the meantime, we shall learn how to program their large and cumbersome ancestors to control an automaton remotely via cable or radio link.

Third, we decided to conduct no extensive research on the subject of visual pattern recognition in this project. This ground rule by no means should be taken as minimizing the importance of the problem of visual perception. On the contrary, it is probably one of the most important problems to be faced in designing automata. But we felt that the perceptual abilities conferred by employing presently existing pattern-recognition methods were more than adequate to permit the use of a real environment sufficiently rich to tax our skills in developing control programs for that environment. In the meantime, research on mechanizing perception could and should continue independently.

Fourth, we decided that the environment of the automaton should be large in extent. Its components may be simple in quality in the beginning, but there should be a non-trivial, extensive environment that the automation is expected to deal with. This ground rule forces us immediately to consider only methods for *efficient* internal representations of the world.*

The eleventh report [11] gave a concise summary of the organization of the Shakey system which can also serve as an overview to the present note:

The robot system is a hierarchical structure in which we shall identify five major levels. Although some of these levels are much more clearly defined than others and some have considerable substructure, the five levels described below constitute a useful division for this exposition. Also, the effectiveness of the system is largely derived from the clear specifications for these levels and their interconnections.

The bottom level of the system consists of the robot vehicle and its connection to the user programs. This connection includes radio and microwave communication links, a PDP-15 peripheral computer and its software, and a communications channel, with its associated software, between the PDP-15 and the PDP-10. This bottom level may be thought of as defining the elementary physical capabilities of the system.

**From [1], pages 1-2.*

The robot vehicle is described in Chapter Two and Appendix A of the present report, and the PDP-15/PDP-10 interface is described in Appendix G of [10].

The heart of the software that controls Shakey is its "model" of the world it inhabits. This model is a global data structure that can be accessed and modified by the other routines. It is described in Chapter Three.

Continuing with the excerpt from [11]:

The second level consists of what we call Low-Level Actions, or "LLAs." These are the lowest-level robot control programs available to user programs in the LISP language, our principal programming tool. The LLAs are programatic handles on the robot's physical capabilities such as "ROLL" and "TILT." They are described in detail in Chapter Four.

So that it can exhibit interesting behavior, our robot system has been equipped with a library of Intermediate-Level Actions, or "ILAs." These third-level elements are preprogrammed packages of LLAs, embedded in a Markov table framework with various perception, control and error-correction features. (Markov formalizations are explained in Chapter Five, Section B.) Each ILA represents built-in expertise in some significant physical capability, such as "PUSH" or "GO TO." The ILAs might be thought of as instinctive abilities of the robot, analogous to such built-in complex animal abilities as "WALK" or "EAT." Chapter Five contains a description of the present set of ILAs, along with the conditions under which they are applicable and how they each can affect the state of the world.

The principal sensor of the perceptual system is the TV camera. Programs for processing picture data have been restricted to a few special "vision" routines, that orient the robot and detect and locate objects. These programs are incorporated into the system at either the ILA or LIA level. The algorithms in these routines are described in Chapter Six and Appendix B.

Above the ILAs we have the fourth level, which is concerned with planning the solutions to problems. The basic planning mechanism is STRIPS, described in Chapter Seven. STRIPS constructs sequences of ILAs needed to carry out specified tasks. Such a sequence, along with its expected effects, can be represented by a triangular table called a

MACROP ("macro operation"). Chapter Eight describes how such MACROPs can be generated in generalized form, thereby enabling an interesting form of learning and plan selection to take place.

Finally, the fifth, or top, level of the system is the executive, the program that actually invokes and monitors executions of the ILAs specified in a MACROP. The current executive program, called PLANEX, is briefly described at the end of Chapter Eight.*

*From [11], pages 3-4.

CHAPTER TWO

The Robot Vehicle, The Computers, and Other Hardware

A. The Vehicle and its Environment

The robot vehicle itself is shown in Figures 1 and 2. It is propelled by two stepping motors independently driving a wheel on either side of the vehicle. It carries a vidicon television camera and optical range-finder in a movable "head." Control logic on board the vehicle routes commands from the computer to the appropriate action sites on the vehicle. In addition to the drive motors, there are motors to control the camera focus and iris settings and the tilt angle of the head. Other computer commands arm or disarm interrupt logic, control power switches and request readings of the status of various registers on the vehicle. Besides the television camera and range-finder sensors, several "cat-whisker" touch-sensors are attached to the vehicle's perimeter. These touch sensors enable the vehicle to know when it bumps into something. Commands from the computer to the vehicle and information from the vehicle to the computer are sent over two special radio links, one for narrow-band telemetering and one for transmission of the TV video from the vehicle to the computer.*

More detailed information about the vehicle can be found in Appendix A at the end of the present report.

The initial environment of the Automaton was real, but contrived. It has been sufficiently simple to allow current visual capabilities to be useful to the Automaton, and sufficiently complex to indicate the weaknesses of current methods and to suggest areas of further research. Perhaps the most important result of our vision-research effort on the Automaton project is an appreciation of the potential complexity of the problem of vision when the real world is the subject matter, and a strong notion that the first step we have taken towards a general capability is very small indeed.

**From [2], page 1.*

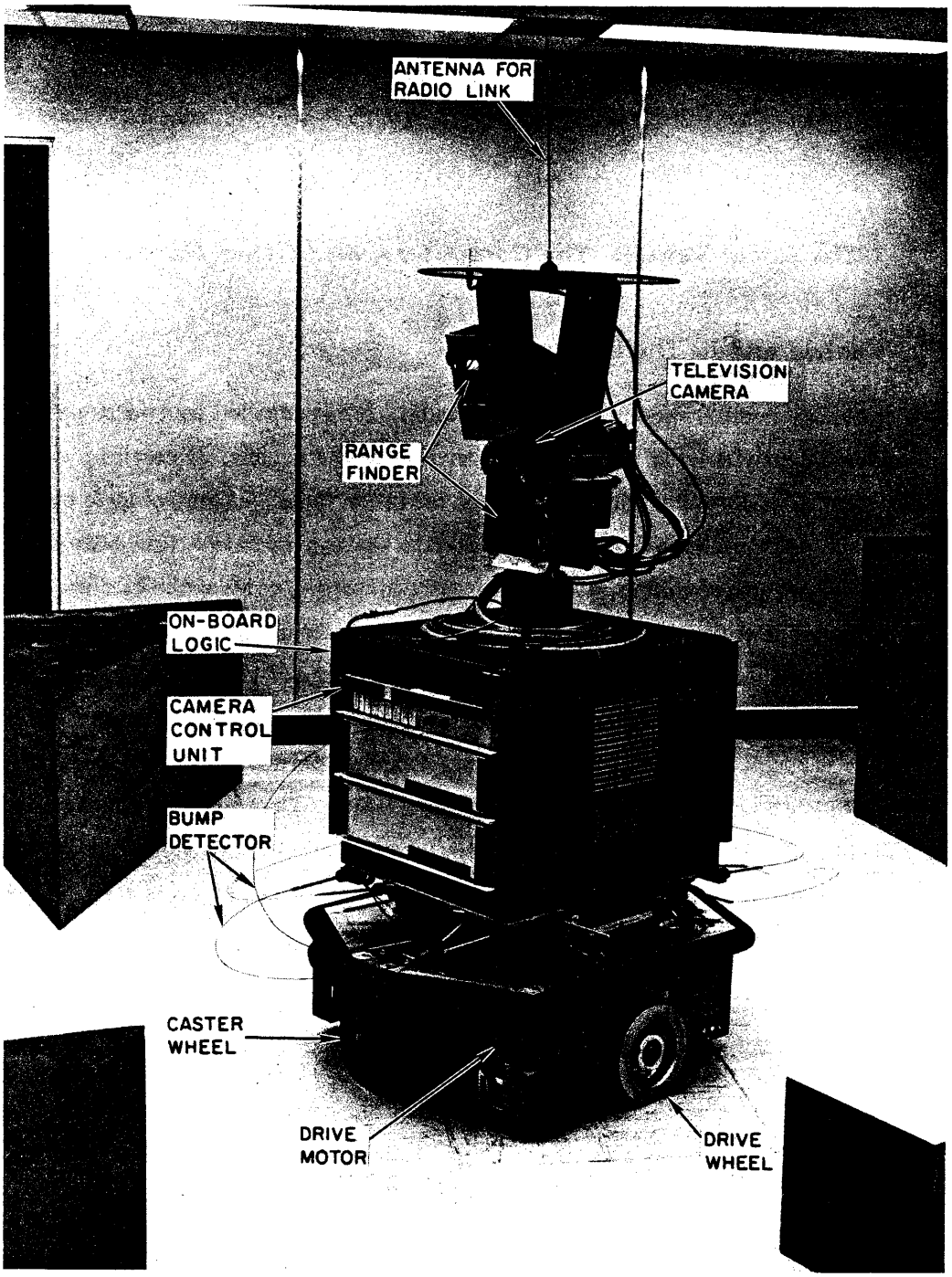


Figure 1: AUTOMATON VEHICLE*

*From [5], page 2.

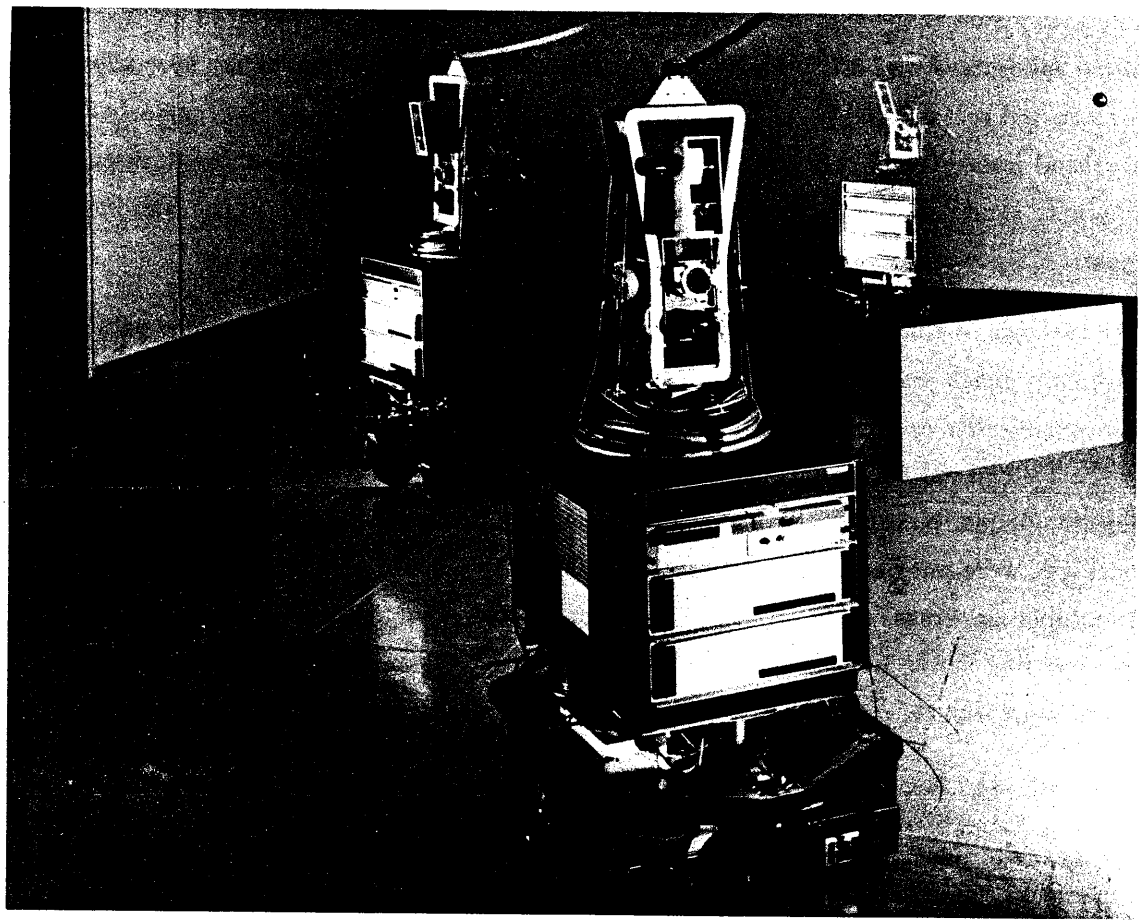


Figure 2: AUTOMATON VEHICLE IN ITS ENVIRONMENT*

**From [5], page 3.*

The current Automaton is restricted by its method of locomotion to move only on nearly flat surfaces. Initially its travel was limited by the length of cable connecting it and the computer. The addition of the radio links allow the Automaton to travel further from the computer room.

The first visual subsystem was designed to specialize in the planar-surfaced environment of our laboratory and office building. The objects in this environment are specially constructed rectangular parallelepipeds and wedges. The use of only the regularly spaced overhead fluorescent lights as well as light colored walls and floor allows us to essentially eliminate shadows and to limit the illumination to a 2-1/2 to 1 range in the computer room.

The surfaces of the objects used are uniformly coated with red, grey, or white paint. Originally black was used to insure high contrast between adjacent surfaces. However, the range-finder relies on reflected light. Red replaced black because it is relatively dark to the TV camera and returns enough light to the range-finder. Thus, not only are the objects opaque, but also have non-specular surfaces. Furthermore no two-dimensional markings were put on the object surfaces. The floor tile was chosen so as not to have any detectable markings. The only two-dimensional marking purposely applied was a dark wall molding at the floor level. The floor has about the same reflectivity as the walls. There were verticle molding strips on one wall which were specular.*

B. Hardware Associated with the Vehicle

An excerpt from [5] describes some of the interface hardware between the vehicle and the SDS computer. Much of this hardware remained unchanged when we substituted a PDP-10 computer for the SDS-940.

Figure 3 shows a block diagram of the hardware system. The system consists of a stationary part interfacing with the SDS 940 computer and the mobile vehicle which is remotely controlled from the fixed equipment via a full duplex radio link. (The data communications interface was described in an Appendix of [4].)

Commands to the vehicle are transmitted in digital form preceded by a module address referring to the module on the vehicle that is expected to act. Each module is equipped

---

**From [5], pages 19-20.*

with its own register. The register holds bits specifying information on desired direction of motion, speed, requested distance, and other special functions. When action is requested, the action starts and continues until completed or interrupted by other control functions in the system. End-of-action or other control interrupts are transmitted back to the stationary equipment in coded form, where they are decoded and sent as interrupts to the computer. Interrupts of a similar nature are ORed together to limit the number of interrupts. Status registers are therefore provided on the vehicle so that status can be interrogated from the computer any time the source of the interrupt is in question.

Special registers for the sensors, such as the range finder, bumpers, etc., are available and can be interrogate by a read operation in the same manner as reading from the module register.

The hardware for the visual system uses the same interface to the computer. The power for the TV camera and the special transmitter for the videodata is controlled from the power-control register on the vehicle. The rest of the visual system is quite independent.

The TV camera consists of one control unit mounted on the platform of the vehicle and one camera head mounted on a pedestal in the center of the vehicle. The camera can be turned ± 180 degrees around a vertical centerline, and it can be titled +60 degrees and -45 degrees around a horizontal axis located below and perpendicular to the optical axis of the camera. The camera is equipped with a manually replaceable lens. The lens mounts in a mechanism with two motors for control of iris and focus. The control of all degrees of freedom of the camera and its lens system is accomplished by stepping motors. The rotation of the camera around the vertical shaft is under control of a servo similar to that used for the wheels of the vehicle. The control from the computer is in the form of LEFT or RIGHT commands of a given number of steps. The camera has one left-rotational terminal switch at +180 degrees rotation and one right-rotational terminal switch at -180 degrees rotation. When these switches close, the rotation in the direction in process is interrupted. The switches also signal the emergency circuit, causing an interrupt signal at the computer. Associated with the shaft rotation, there is also a pan distance counter. The content of the counter can be transmitted to the computer. The tilt of the camera is controlled by a stepping motor operated at a constant step rate. The motor reacts to a TILT UP or TILT DOWN command for a given number of steps. The tilt mechanism has limiting switches up and down. The limit switches stop the tilt and signal the interrupt circuits in the computer. The content of the tilt counter can be transmitted to the

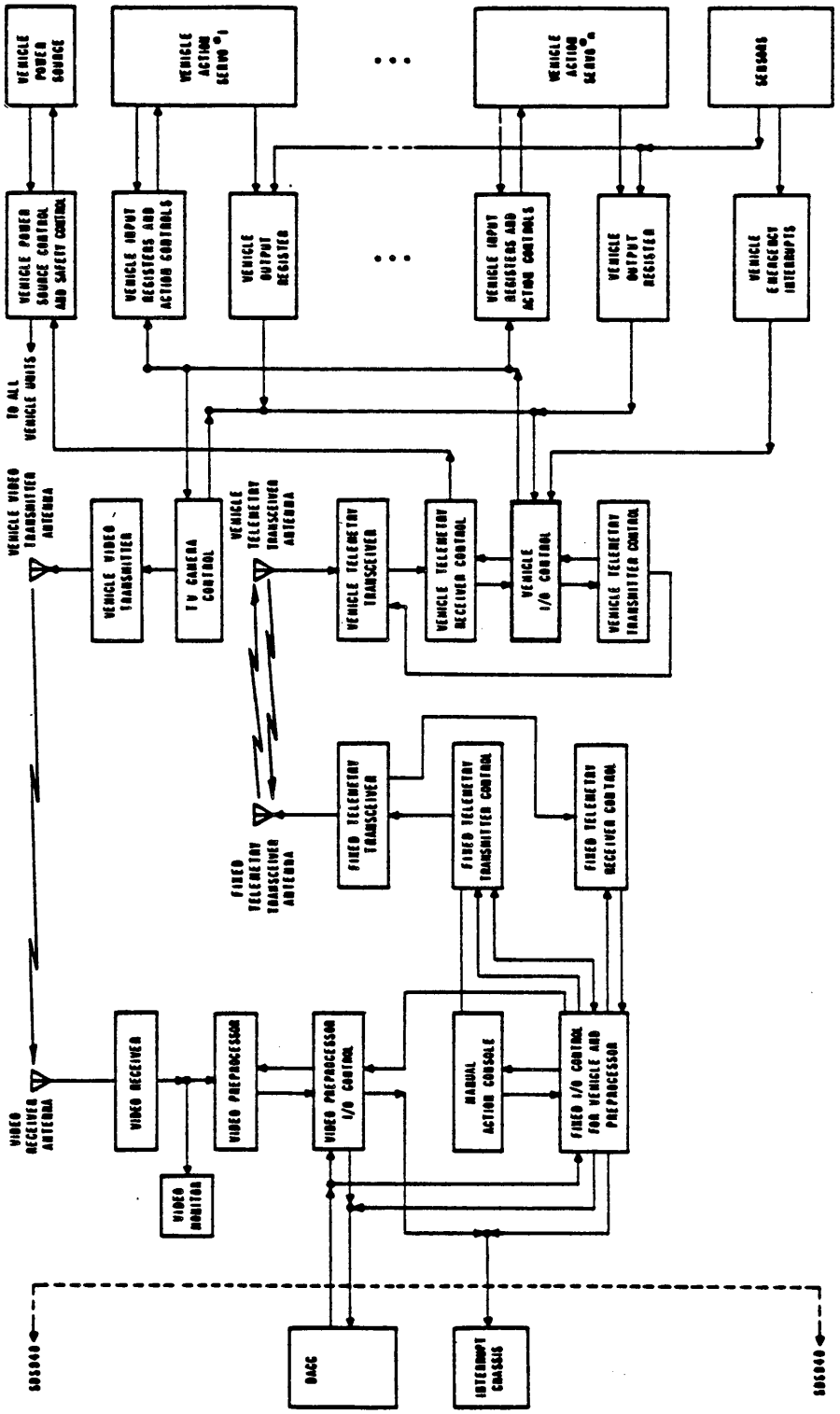


Figure 3: AUTOMATON-SYSTEM BLOCK DIAGRAM*

**From [5], page 30.*

computer. A brake mechanism locks the camera in its tilt position when power is removed from the motor.

Only one lens is presently used. Focus is controlled by one stepping motors and iris by another. The rotation is limited by limit switches. The limit switches preset the counters at maximum focus and minimum iris associated with the stepping motors.

The control logic has an up-down counter for distance and direction.*

C. The Computer System

The Artificial Intelligence Group computer complex consists of the following parts:

- PDP-10 computer and peripherals
- PDP-15 computer and peripherals (including the robot)
- An interprocessor buffer to connect the two computers.

These are interconnected as shown in Figure 4.

The PDP-10 system has 192K (K = 1024) words of 36-bit memory. 32K is DEC MD10 memory. The rest is Ampex RG10 memory, consisting of one 32K memory with interface and one 128K memory interface and four modules of 32K each. All memory has four ports. These are occupied by:

- PDP-1: central processor
- DF10 data channel
- Bryant drum controller
- DA25C interface.

The Bryant drum is a high-speed autolift drum which has a 1.5-million-word capacity. It is planned that it will be used for swapping and some system files. The drum controller interfaces directly into the memory rather than going through a data channel.

**From [5], pages 29-32.*

The DF10 data channel is used to handle I/O from two peripherals: the disk pack drives and the TV A/D converter.

The interface between the disk pack drives and the DF10 data channel was built by Interactive Data Systems, Inc.

The disk pack drives are manufactured by Century Data Systems and handle the 20-surface disk packs. This means that each disk pack has a 5-million-word capacity. The packs themselves are manufactured by Caelus Inc. The disk pack system is used as secondary storage.

Currently, we are also using one disk pack drive as a swapping device for the time-sharing system.

The TV A/D converter is an SRI-designed and -built device. It handles data from the robot TV camera at a rate of one word every 1.5 microseconds. It is capable of processing either 120X120 or 240X240 pictures with 32 levels of gray scale.

The DA25C is the PDP-10 side of the interprocessor buffer. It handles data at one 36-bit word every 8 microseconds. We have programmed it such that the PDP-10 is always in control and can interrupt any transmission in order to initiate one of its own.

The DA25D is the PDP-15 side of the interprocessor buffer. Each PDP-10 word is split into two PDP-15 words (18 bits each). It also does the reverse operation. It operates on the PDP-15 I/O bus as a single-cycle device; however, its internal logic uses three cycles per word.

The PDP-15 has 12K of core memory and an I/O processor. All devices are "daisy chained" on the I/O bus. These include an Adage display, paper tape, DEC tape, A/D converter, D/A converter, ARPA network IMP, and the SRI robot.

The Adage display provides a high-speed graphics capability. It will be refreshed from the PDP-15 core. The display lists will be prepared in the PDP-10 and executed from the PDP-15. Capabilities include incremental mode, print mode, dotted lines, and intensity control.*

A special software interface was also written for use on the PDP-10

**From [9], pages 15-16.*

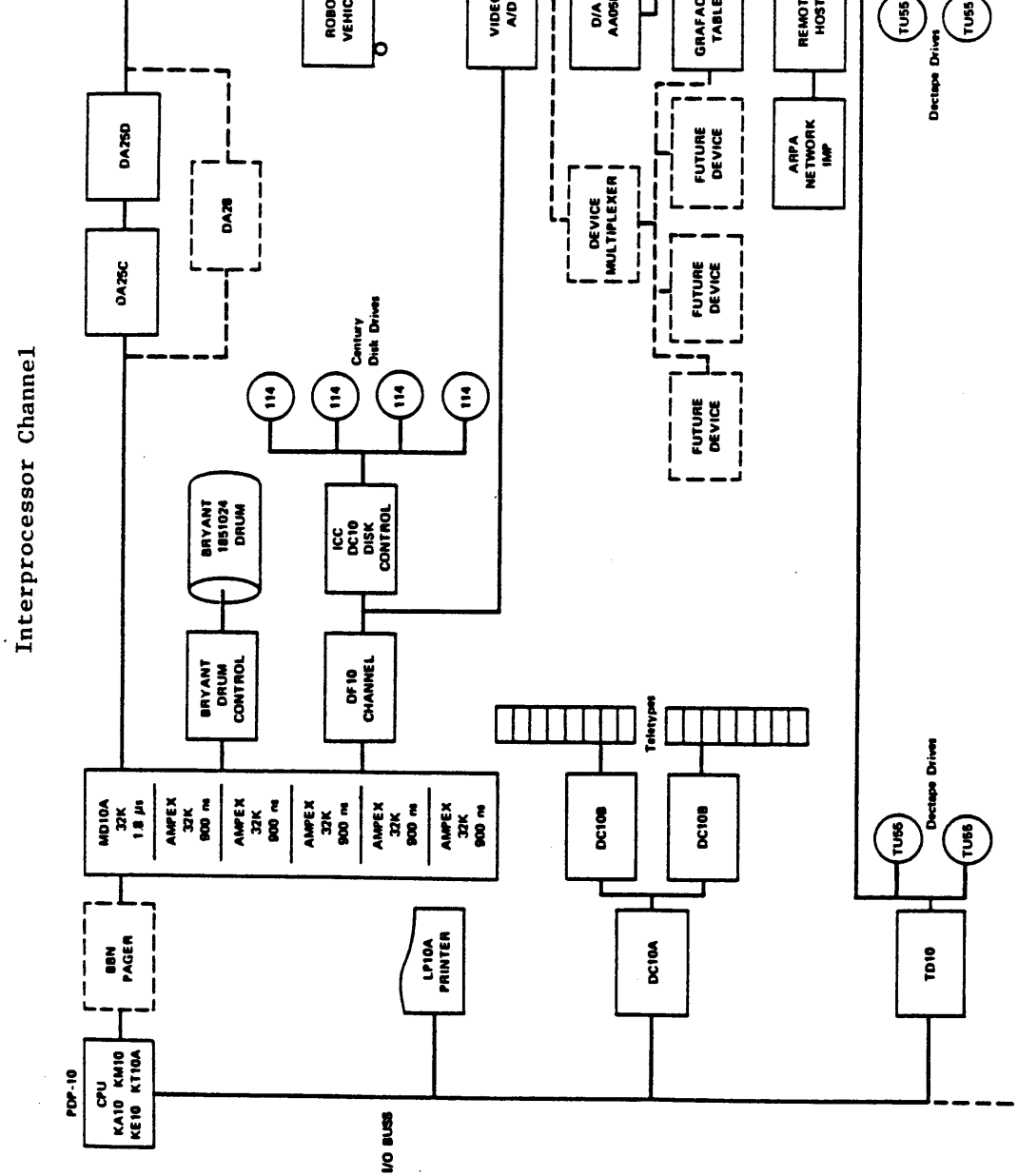


Figure 4: SRI ARTIFICIAL INTELLIGENCE GROUP COMPUTER SYSTEM*

*From [10], page 50.

computer to allow FORTRAN (or FORTRAN-compatible MACRO) subroutines and functions to be run under the LISP operating system. This interface is described in [13].

CHAPTER THREE

Shakey's Model of the World

A. The Robot's World Model

As a result of our experience with the previous robot system (i.e., the one using the SDS-940) and our desire to expand the robot's experimental environment to include several rooms with their connecting hallways, we have adopted new conventions for representing the robot's model of the world. In particular, whereas the previous system had the burden of maintaining two separate world models (i.e., a map-like grid model and an axiom model), the new system uses a single model for all its operations (an axiom model); also, in the new system conventions have been established for representing doors, wall faces, rooms, objects, and the robot's status.

The model in the new system is a collection of predicate calculus statements stored as prenexed clauses in an indexed data structure. The storage format allows the model to be used without modification as the axiom set for STRIPS' planning operations (Chapter Seven) and for QA3.5's theorem-proving activities [14, 15].

Although the system allows any predicate calculus statement to be included in the model, most of the model will consist of unit clauses (i.e., consisting of a single literal) as shown in Table 1. Nonunit clauses typically occur in the model to represent disjunctions (e.g., box2 is either in room K2 or room K4) and to state general properties of the world (e.g., for all locations loc1 and loc2 and for all objects ob1, if ob1 is at location loc1 and loc1 is not the same location as loc2, then ob1 is not at location loc2).

We have defined for the model the following five classes of entities: doors, wall faces, rooms, objects, and the robot. For each of these classes we have defined a set of *primitive* predicates which are to be used to describe these entities in the model. Table 1 lists these primitive predicates and indicates how they will appear in the model. All distances and locations are given in feet and all angles are given in degrees. These quantities are measured with respect to a rectangular coordinate system oriented so that all wall faces are parallel to one of the X-Y axes. The NAME predicate associated with

each entity allows a person to use names natural to him (e.g., halldoor, leftface, K2090, etc.) rather than the less-intuitive system-generated names (e.g., d1, f203, r4450, etc.).

Figure 5 shows a sample environment and a portion of the corresponding world model. Rooms are defined as any rectangular area, and therefore, the hallway on the left is modeled as a room. There is associated with each room a grid structure that indicates which portions of the room's floor area have not yet been explored by the robot. During route planning the grid is employed to help determine if a proposed path is known blocked, known clear, or unknown.

Four wall faces are modeled in Figure 5. The FACELOC model entry for each face indicates the face's location on either the X or Y coordinate depending on the face's orientation. There is associated with each face a grid structure to indicate which portions of the wall face have not yet been explored by the robot. This grid is used in searching wall faces for doors and signs.

Two doors are modeled in Figure 5. The DOORLOC model entry for each door indicates the locations of the door's boundaries on either the X or Y coordinate, depending on the orientation of the wall in which the door lies. Any opening between adjoining rooms is modeled as a door, so that the complete model of the environment diagrammed in Figure 5 would have a door connecting rooms R1 and R3. This door coincides with the south face of room R3 and will always have the status "open."

The RADIUS and AT model entries for the object modeled in Figure 5 define a circle circumscribing the object. These entries simplify the route-planning routines by allowing each object to be considered circular in shape. Our current set of primitive predicates for describing objects is purposely incomplete; we will add new predicates to the set as the need for them arises in our experiments.

We do not wish to restrict the model to only statements containing primitive predicates. The motivation for defining such a predicate class is to restrict the domain of model entries that the robot action routines have responsibility for updating. That is, it is clear that the action routine that moves the robot must update the robot's location in the model, but what else should it have to update? The model may contain many other entries whose validity depends on the robot's previous location (e.g., a statement indicating that the robot is next to some object), and the system must be able to determine that these statements may no longer be valid after the robot's location has changed.

We have responded to this problem by assigning to the action routines (discussed in Chapters Four and Five) the responsibility for updating only those model statements which are unit clauses and contain a primitive predicate. All other statements in the model will have associated with them the primitive predicate unit clauses on which their validity depends. When such a nonprimitive statement is fetched from the model, a test will be made to determine whether each of the primitive statements on which it depends is still in the model; if not, then the nonprimitive statement is considered invalid and is deleted from the model. This scheme ensures that new predicates can be easily added to the system and that existing action routines produce valid models when they are executed.

B. Model-Manipulating Functions

We have designed and programmed a set of LISP functions for interacting with the world model. These functions are used both by the experimenter (as he defines and interrogates the model) and by other routines in the system to modify the model. To the experimenter at a teletype, these functions are accessible as a set of commands. A brief description of these commands follows.

ASSERT This is the basic command for entering new axioms into the model. The user follows the word ASSERT by either CUR or ALL to indicate whether the entries are to be for the current model or are to be considered part of all models. The system then prompts the user for predicate calculus statements to be typed in using the QA3.5 expression input language. After each statement is entered, the system responds with "OK" and requests the next statement. To exit the ASSERT mode the user types "↑."

FETCH This is the basic command for model queries. The user follows the word FETCH by an atom form, and the system types out a list of all unit clauses in the model that match the form. Each term in an atom form is either a constant or a dollar sign. The dollar sign indicates an "I don't care" term and will match anything. The last term of an atom form can also be the characters "$*" to indicate an arbitrary number of "I don't care" terms. For example, the atom form "(AT ROBOT $*)" will fetch the location of the robot, and the atom form "(INROOM $ R1)" will fetch a list of model entries indicating each of the objects in room R1.

DELETE This is the basic command for removing statements from the model. The user follows the word DELETE by an atom form, and the system deletes all unit clauses in the model that match the form. Atom forms have the same syntax and semantics for the DELETE command as described above for the FETCH command.

REPLACE This is a hybrid command combining the operations of DELETE and ASSERT. The user follows the word REPLACE by an atom form and by a predicate calculus statement. The system first deletes all unit clauses in the model matching the atom form and then enters the statement into the model. This command is useful for operations such as changing the robot's position in the model, indicating in the model that a previously closed door is now open, and so forth.*

**From [10], pages 9-15.*

PRIMITIVE PREDICATES FOR THE ROBOT'S WORLD MODEL

Primitive Predicate	Literal Form	Example Literal
FACES		
type	type(face"face")	type(f1 face)
name	name(face name)	name(f1 leftface)
faceloc	faceloc(face number)	faceloc(f1 6.1)
grid	grid(face grid)	grid(f1 g1)
boundsroom	boundsroom(face room direction)	boundsroom(f1 r1 east)
DOORS		
type	type(door"door")	type(d1 door)
name	name(door name)	name(d1 halldoor)
doorlocs	doorlocs(door number number)	doorlocs(d1 3.1 6.2)
joinsfaces	joinsfaces(door face face)	joinsfaces(d1 f1 f2)
joinsrooms	joinsrooms(door room room)	joinsrooms(d1 r1 r2)
doorstatus	doorstatus(door status)	doorstatus(d1 "open")
ROOMS		
type	type(room"room")	type(r1 room)
name	name(room name)	name(r1 K29090)
grid	grid(room grid)	grid(r1 g1)
OBJECTS		
type	type(object"object")	type(o1 object)
name	name(object name)	name(o1 box1)
at	at(object number number	at(o1 3.1 5.2)
inroom	inroom(object room)	inroom(o1 r1)
shape	shape(object shape)	shape(o1 wedge)
radius	radius(object number)	radius(o1 3.1)
ROBOT		
type	type("robot""robot")	type(robot robot)
name	name("robot"name)	name(robot shakey)
at	at("robot" number number)	at(robot 4.1 7.2)
theta	theta("robot"number)	theta(robot 90.1)
tilt	tilt("robot"number)	tilt(robot 15.2)
pan	pan("robot"number)	pan(robot 45.3)
whiskers	whiskers("robot"integer)	whiskers(robot 5)
iris	iris("robot"integer)	iris(robot 1)
override	override("robot"integer)	override(robot 0)
range	range("robot"number)	range(robot 30.4)
tvmode	tvmode("robot"integer)	tvmode(robot 0)
focus	focus("robot"number)	focus(robot 30.7)

Table 1: PRIMITIVE PREDICATES FOR THE ROBOT'S WORLD MODEL*

*From [10], Page 11.

ROOMS

type(r1 room)
name(r1 mainroom)
grid(r1 g1)

type(r2 room)
name(r2 office)
grid(r2 g2)

type(r3 room)
name(r3 hall)
grid(r3 g3)

FACES

type(f1 face)
name(f1 nfrl)
faceloc(f1 15.0)
grid(f1 g4)
boundsroom(f1 r1 north)

type(f2 face)
name(f2 sfr2)
faceloc(f2 15.5)
grid(f2 g5)
boundsroom(f2 r2 south)

type(f3 face)
name(f3 wfr2)
faceloc(f3 5.0)
grid(f3 g6)
boundsroom(f3 r2 west)

DOORS

type(d1 door)
name(d1 officedoor)
doorlocs(d1 10.0 12.5)
joinsfaces(d1 f1 f2)
joinsrooms(d1 r1 r2)
doorstatus(d1 open)

type(d2 door)
name(d2 halldoor)
doorlocs(d2 22.5 25.0)
joinsfaces(d2 f4 f3)
joinsrooms(d1 r3 r2)
doorstatus(d2 closed)

OBJECTS

type(o1 object)
name(o1 box)
at(o1 14.1 20.3)
inroom(o1 r2)
shape(o1 rectangular)
radius(o1 1.5)

ROBOT

type(robot robot)
name(robot shakey)
at(robot 7.3 10.4)
theta(robot 90)

.
.
.

Figure 5: EXAMPLE MODEL*

*From [10], page 13.

CHAPTER FOUR

The Low-Level Actions

A. Introduction

The low-level actions, or "LLAs," define the interface between major robot software packages and the bottom, hardware-oriented level of the system. The intermediate-level actions (ILAs), to be described in Chapter Five, control the operation of these LLAs. The LLAs, in turn, communicate with the PDP-15 computer and the robot vehicle according to the protocol described in Appendix G of [9].

In this section we shall describe the upper face of the LLAs, i.e., the face presented to higher-level programs.

Since the robot moves very slowly, we have taken great pains to permit the user to view the robot as behaving asynchronously to as great an extent as appropriate. Thus, the user must take cognizance of this asynchrony by confirming the completion of "settling" on any robot activity before doing anything that assumes that activity to have been successful. This low-level software package provides the necessary interlocking in the following manner. Communications between the user and the robot are separated into two unidirectional channels: orders from the user to the robot are handled by calls on LLAs (i.e., the functions in this package); the current state of the robot's world is reflected in the robot's world model. Now, the functions by which the user can access these particular entries in the robot's world model have special provisions to ensure that an activity has settled before granting access to any part of the model which that activity might affect. For example, one might move the robot to a given location by first turning it to face the target spot and then rolling it straight forward by the required distance. One could conceivably confirm the initial turn (by interrogating the proper part of the

This sort of synchronization is effected in another circumstance having to do with interlocks between activities. In particular, each activity has associated with it certain conflicting activities. (For example, one cannot take a TV picture while the robot's head is panning.) A set of initiation functions automatically take cognizance of all possible conflicts: each ensures that all potentially conflicting activities are settled before initiating its own activity. For the purpose of programming actual use of the robot, however, one should note that settling of an activity does not necessarily mean its successful completion. For example, a roll can terminate by the robot unexpectedly bumping into some obstacle—this will "settle" the roll, but the robot cannot be assumed to have attained its destination.

B. Measurement and Control

Before proceeding further, we shall define the precise robot capabilities that the LLAs control. Shakey can move about the floor by turning his body and by rolling straight forward or backward, and he can pan and tilt his head. He can take pictures and range-finder readings, and he can adjust the focus and iris states of the TV camera's lens. Finally, he can set some global parameters both for taking TV pictures and for rolling or turning. These ten activities will be more fully explained below. First we shall describe the measurement conventions in Shakey's environment.

Angles are measured in degrees, and we will call the principal value of an angle that value between -180° and +180°. The bearing of the robot is a horizontal angle referred to the positive direction of the global y-axis; thus the robot is parallel to the x-axis in the negative sense when its bearing is 90°. The pan angle of the robot's head is a horizontal angle referred to the robot's bearing, and the tilt angle of the robot's head is a vertical angle measured from the horizontal plane. Thus, when the robot has its pan angle at zero and the tilt angle at-45°, the range-finder and TV camera are pointed at the floor right before its very wheels.

We turn now to optical values. The iris of the TV camera is set in exposure value units (EVs), which have a logarithmic relation to f-numbers: increasing the EV number by one doubles the amount of light arriving at the inner regions of the TV camera. Focus values and range-finder readings are distances in feet from the intersection of the axes about which the robot's head tilts and pans. That point in turn is about 4 feet 1-1/2 inches above the floor and 9 inches forward of the axis about which the robot turns, when the robot is standing (or sitting or whatever it does) on a level flat floor.

Having covered the numeric quantities in the robot's world, we have but a few other items to discuss. Perhaps the simplest of these to describe is a TV picture: it resides on a disk file in FORTRAN binary format. Now TV pictures are digitized in square arrays of picture elements; the size of the array is constant, but one can select two coarsenesses: 120 or 240 picture elements on a side. One can, however, alter the configuration of the array for the sake of special stereo optics. These two options are combined into one number called the tvmode, as follows:

"tvmode: 0 means 120 X 120 nonstereo
"tvmode" 1 means 120 X 120 stereo
"tvmode" 2 means 240 X 240 nonstereo
"tvmode" 3 means 240 X 240 stereo.

To explain the last two quantities of this section, we must first explain the two main tactile sensors of the robot and how they interact with the roll and turn activities. The tactile sensors are seven catwhiskers and a pushbar; each catwhisker can signal engagement with an obstacle, and the pushbar can signal each of two levels of pressure: mere engagement and hard contact. All nine of these conditions are reflected in a quantity called the whiskerword; to a first approximation each of these conditions has its own bit in the whiskerword, whose format is shown in the following table:

Bit No.	Octal Code	Meaning of "1"
21	040000	Pushbar is engaged and ready to push.
23	010000	Left front whisker is engaged.
25	002000	Front horizontal whisker is engaged.
26	001000	Right front whisker is engaged.
28	000200	Right rear whisker is engaged.
29	000100	Encountered immovable object and backed off.
30	000040	Rear whisker is engaged.
33	000004	Left rear whisker is engaged.
35	000001	Front vertical whisker is engaged.

The robot has a couple of motor reflexes pertinent to this discussion: it will stop moving whenever the pushbar becomes disengaged, and it will not move while a catwhisker is

engaged. However, these two reflexes can be overridden selectively; the corresponding orders are sent to the PDP-15 by means of the override activity and the override code word, which has the following significance:

Code Word	Pushbar	Catwhisker
0	Enabled	Enabled
1	Enabled	Overridden
2	Overridden	Enabled
3	Overridden	Overridden

C. The LLA Portion of Shakey's Model

We will now enumerate and define the 17 predicates by which the robot's lowest-level state is represented in the axiomatic world model. They are:

Atom in Axiomatic Model	Affected By
(AT ROBOT xfeet yfeet)	ROLL
(DAT ROBOT dxfeet dyfeet)	ROLL
(THETA ROBOT degreesleftofy)	TURN
(DTHETA ROBOT dthetadegrees)	TURN
(WHISKERS ROBOT whiskerword)	ROLL, TURN
(OVRID ROBOT overrides)	OVRID
(TILT ROBOT degreesup)	TILT
(DTILT ROBOT ddegreesup)	TILT
(PAN ROBOT degreesleft)	PAN
(DPAN ROBOT ddegreesleft)	PAN
(IRIS ROBOT evs)	IRIS
(DIRIS ROBOT devs)	IRIS
(FOCUS ROBOT feet)	FOCUS
(DFOCUS ROBOT dfeet)	FOCUS
(RANGE ROBOT feet)	RANGE
(TVMODE ROBOT tvmode)	TVMODE
(PICTURESTAKEN ROBOT ±picturestaken)	SHOOT

The two predicates AT and THETA give the position and bearing of the robot itself in the global coordinate system; the statistical uncertainties are given by the predicates DAT and DTHETA, which are separated from AT and THETA to facilitate planning. The state of the whiskerword is updated whenever a ROLL or TURN settles, and the OVRID predicate reflects the state of the overrides in the robot.

The TILT and PAN predicates refer to the direction the robot's head is pointed. DTILT and DPAN give corresponding error estimates. All three angles (tilt angle, pan angle, and heading THETA) are stored as their principal values. RANGE gives the value resulting from the *most recent* range-finder reading. The PICTURESTAKEN predicate, which we will describe more fully in our discussion of the SHOOT activities, gives the approximate number of pictures taken to date. The meanings of the rest of the predicates should be clear from the previous discussion.

D. The LLAs

The predicates are the means by which the robot tells the user about its state; the LLAs provide the means by which the user tells the robot to alter its state. One should understand that this clean division is largely just formal; in practice an interrogation of a predicate is intercepted by a function that ensures settling of any relevant robot activities before proceeding to the actual access. Also, the initiation of an action does not guarantee its completion; actions may terminate for a variety of reasons, such as engagement of limit switches or malfunctions in the telemetry link. The state of the system after an action may be determined by investigating the model.

The following functions initiate fundamental low-level activities (whenever numeric parameters are used, negative numbers are permissible and mean motion in the direction opposite to that indicated):

TILT degreesup tilts the robot's head upward by "degreesup" degrees. The motion can be prematurely terminated by a limit switch.

PAN degreesleft pans the robot's head by "degreesleft" degrees to the left or far enough to activate a limit switch.

FOCUS feetout the TV camera is initially focused on a plane removed by some focal distance from the center of the head's gimbals; this function increases that distance by "feetout" feet. Of course the range of focal distances is limited by limit switches.

IRIS evs opens the robot's iris (on the TV camera) by "evs" EVs. Thus if "evs" has the value 1, this form will double the amount of light getting into the TV camera. There are limits for this activity too.

OVRID overrides set the overrides as specified by the "overrides" code work.

TVMODE tvmode sets the TV mode as specified by the "tvmode" code word.

RANGE reads the robot's range-finder; this automatically includes turning on the range-finder and waiting for it to warm up.

SHOOT puts a TV picture onto the disk file "TV.DAT." The picture is taken according to the current TV mode. Assuming correct operation of hardware and software, a subsequent examination of the PICTURESTAKEN atom (in the world model) will yield a positive integer giving the number of current pictures in a series (1, 2, 3,...) begun when the robot system was loaded or initialized. In the event of an unrecovered system malfunction (e.g., transmission error), the value stored with PICTURESTAKEN will be the negative of the serial number of the last successfully taken picture.

ROLL feet tells the robot to roll forward by "feet" feet. This activity has three normal ways of prematurely terminating: the robot can come into contact with an obstacle, engaging a catwhisker; it can lose contact with an object it is pushing, disengaging the pushbar; or it can encounter an immovable object, causing the pushbar to come on hard. The first two conditions cause the robot to stop by reflex actions that can be overridden; the last causes the robot to attempt to free itself using more complex evasive actions in a reflex that cannot be overridden. When the robot encounters an immovable object, it will not only stop, but it will back away from it by some distance, currently a constant 6 inches. (Of course, the information in the model will be correctly maintained.) The whiskerword in the model is updated at the end of a ROLL or TURN; it contains the description of the current state if the catwhiskers and pushbar are returned from the robot, but it has another bit for immovable objects—this bit showing the history of an event rather than showing a current state. This bit is set only when the whiskerword is updated the first time after hard contact.

TURN degreesleft tells the robot to turn to the left by "degreesleft" degrees. Otherwise the above description of the ROLL activity applies excepting only the way immovable objects are evaded. In this case, the robot turns back; currently it turns back to its initial heading.

The functions discussed so far that initiate motions have been incremental in form if not in essence. However, even this level of robot software has a memory of the various aspects of the robot's position in the axiomatic model so dutifully maintained by the settling functions. Capitalizing on this circumstance, we have also provided some functions to initiate motions to a given goal (rather than by a given amount). Although these functions are formally and conceptually outside the lowest LISP level of robot software, they have sufficiently simple internal structure that it is convenient to describe them here rather than in the next (ILA) chapter. With one exception we expect their meanings to be self-evident. These additional initiation functions are:

(TILTO degreesup)
(PANTO degreesleft)
(FOCUSTO feet)
(IRISTO evs)
(ROLLTO xfeet yfeet)
(TURNTO degreeslefttofy).

The exception is ROLLTO: it must first turn the robot to point toward its goal, so it must do (and does) more than simple calling the corresponding incremental function with the difference between the desired and current position.

E. Summary

Table 2 is a summary of Shakey's low-level activities. Figure 6 sketches how these activities fit into the overall system control structure.*

From [11], pages 25-33.

Table 2**

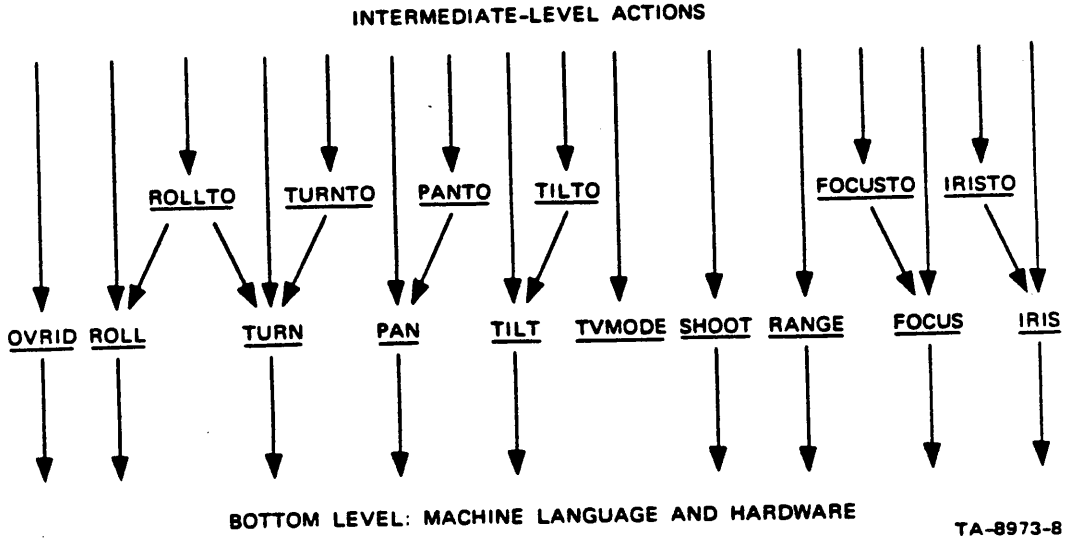
LOW-LEVEL ACTIVITIES OF ROBOT

Initiation Functions		Conflicts (+self)	Terminating Conditions	Needs fi
Primary	Absolute			
(TILT degreesup)	(TILTO degreesup)	RANGE,SHOOT	upper limit (35°) lower limit (-45°)	TILT,DTILT
(PAN degreesleft)	(PANTO degreesleft)	RANGE,SHOOT	left limit (116°) right limit (-107°)	PAN,DPAN
(FOCUS feetout)	(FOCUSTO feetout)	SHOOT	near limit far limit	FOCUS,DFOC
(IRIS evs)	(IRISTO evs)	SHOOT	open limit closed limit	IRIS,DIRIS
(OVRID overrides)	--	TURN,ROLL	--	-
(TVMODE tvmode)	--	SHOOT	--	-
(RANGE)	--	TURN,ROLL,TILT,PAN	--	-
(SHOOT)	--	TVMODE,ROLL,TURN, TILT,PAN,IRIS,FOCUS	--	TVMODE,PIC
(ROLL feet)	(ROLLTO xfeet yfeet)*	TURN,RANGE,OVRID, SHOOT,ROLL	bump-ignored bump-stopped drop object-stopped immovable object-backed off	AT,DAT,THE
(TURN degreesleft)	(TURNTO degreesleft)			THETA,DTHE

*ROLLTO evokes the TURN activity.

Table 2: LOW-LEVEL ACTIVITIES OF ROBOT**

**From [11], page 84.


INTERMEDIATE-LEVEL ACTIONS
ROLLTO
TURNTO
PANTO
TILTO
FOCUSTO
IRISTO
OVRID
ROLL
TURN
PAN
TILT
TVMODE
SHOOT
RANGE
FOCUS
IRIS
BOTTOM LEVEL: MACHINE LANGUAGE AND HARDWARE

TA-8973-8

CHAPTER FIVE

The Intermediate-Level Actions

The intermediate-level actions (ILAs) are described in excerpts from two reports [10 and 11]. Each excerpt is more-or-less self contained (and thus some redundant material is reprinted), but both should be read for a complete picture. The first excerpt discusses early plans for the ILAs:

A. Introduction

As with most programming tasks, the problem of programming robot actions is simplified when it is done in terms of well-defined subroutines. At the lowest level it is natural to define routines that have a direct correspondence with low-level robot actions—routines for rolling, turning, panning, taking a range reading, taking a television picture, and so forth. However, these routines are too primitive for high-level problem solving. Here it is desirable to assume the existence of programs that can carry out tasks such as going to a specified place or pushing an object from one place to another. These intermediate-level actions (ILAs) may possess some limited problem-solving capacity, such as the ability to plan routes and recover from certain errors, but the ILAs are basically specialized subroutines. None of these routines has as yet been written. However, considerable thought has been devoted to their design, and this section describes our plans for a set of ILAs that are suitable for use with the STRIPS problem-solving system.

Perhaps the most difficult problem that confronts the designer of ILAs is the problem of detecting and recovering from errors. Sometimes errors are detected automatically, as when an interrupt from a touch sensor indicates the presence of an unexpected obstacle. Other times it is necessary to make explicit checks, such as checking to be sure that a door is open before moving through it. When an error is detected, the problem of recovery arises. This problem can be very difficult, and is one aspect that distinguishes work in robotry from other work in artificial intelligence.

It is natural to think of an intermediate-level action as a composition of somewhat lower-level actions, which in turn are compositions of lower-level actions. While this

hierarchical organization possesses many advantages (and it is in fact the organization that we use), it is not ideally suited for error recovery. Errors are made most frequently at low levels by routines that are too primitive to cope with them. An error message may have to be passed up through several levels of routines before reaching one possessing sufficient knowledge of both the world and the goal to take corrective action. If any routine can fail in several ways, this presents the highest-level routine with a bewildering variety of error messages to analyze, and requires explicit coding for a large number of contingencies.

To circumvent this problem, we have chosen to have the subroutines communicate through the model. With a few special exceptions, neither answers nor error messages are explicitly returned by subroutines. Instead, each routine uses the information it gains to update the model. It is the responsibility of the calling routine to check the model to be sure that conditions are correct before taking the next step in a sequence of actions. Detection of an error causes returns through the sequence of calling programs until the routine that is prepared to handle that kind of error is reached. In the following sections we describe in more detail the formal mechanism by which this is done.

B. The Markov Algorithm Formalization

1. General Considerations

The formal structure of each ILA routine is basically that of a Markov algorithm.* Each routine is a sequence of statements. Each statement consists of a statement label, a predicate, an action, and a control label. When a routine is called, the predicates are evaluated in sequence until one is found that is satisfied by the current model. Then the corresponding action is executed. The control label indicates a transfer of control, either to another labeled statement or to the calling routine.

Table 3 gives a specific example of an ILA coded in this form. This routine, gotoadjroom (room1, door, room2), is intended to move the robot from room1 to room2 through the specified door. The first test made is a check to be sure that the robot is in room1. If it is not, an error has occurred somewhere. Since this routine is not prepared to handle that kind of error, no action is taken, and control is returned to the calling routine. The subroutine return is indicated by the "R" in the control field.

*It also bears a close resemblance to Floyd-Evans productions.

Under normal circumstances, the first two predicates will be false. The third predicate is always true, and the corresponding action sets the value of a local variable "s" to give the status of the door. The function "doorstatus" computes this from the model, and evaluates to either OPEN, CLOSED, or UNKNOWN. Rather than tracing through all of the possibilities, let us consider a normal case in which the door is open but the robot is neither in front of nor near it. It this case, the action taken is the last one, navto(nearpoint(room1,door)). Here the function "nearpoint" computes a goal location near the door. The function "navto" is another ILA that plans a route to the goal point and eventually executes a series of turns and rolls to get the robot to that goal. Of course, unexpected problems may prevent the robot from reaching that goal. Nevertheless, whether navto succeeds or fails, when it returns to gotoadjroom the next predicate checked will be that of statement 4. If navto succeeds and the robot is actually in front of the door, the bumblethru routine will be called to get the robot into room2. If navto had failed and the robot is not even near the door, navto will be tried again. Clearly, this exposes the danger of being trapped in fruitless infinite loops. We shall describe some simple ways of circumventing this problem shortly.

SUBROUTINE GOTOADJROOM(ROOM1,DOOR,ROOM2)

Label	Predicate	Action	Control
1	~ in(room1)		R
2	in(room2)		R
3	T	setq(s,doorstatus(door))	4
4	infrontof(door)∧eq(s,OPEN)	bumblethru(room1,door,room2)	2
	near(door)∧eq(s,OPEN)	align(room1,door,room2)	4
	near(door)∧eq(s,UNKNOWN)	doorpic(door)	3
	eq(s,CLOSED)		R
	T	navto(nearpt(room1,door))	4

Table 3: SUBROUTINE GOTOADJROOM (ROOM1,DOOR,ROOM2)

2. Predicates and Actions

The predicates used in the ILAs have the responsibility of seeing that preconditions for an action are satisfied. In general, the evaluation of predicates is based on information contained in the model. If this information is incorrect, the resulting action will usually be inappropriate. However, the act of taking such an action will frequently expose errors in the model. When the model is updated (which typically occurs after bumping into an object or analyzing a picture), the values of predicates can and do change. Thus, the values of the predicates will depend on the way the execution of the ILA proceeds, and will steer the routine into (hopefully) appropriate actions when errors are encountered.

The actions can be any executable program. The most common actions are to compute the values of local variables, update the model, call picture-taking routines that update the model, or call other ILAs. Only the first of these causes any answers to be returned directly to the calling program. This constraint of communicating through the model occasionally leads to computational inefficiencies. For example, the very computation used by one routine to determine that it has completed its job successfully may be repeated by the calling routine to be sure that the job has been done. While some of these inefficiencies could be eliminated with modest effort, they appear to be of minor importance compared to the value of having a straightforward solution to the problem of error recovery.

3. Loop Suppression

We mentioned earlier that the failure of a lower-level ILA might result in no changes in the model that are detected by the calling ILA. In this case, one can become trapped in an infinite loop. There are a number of ways to circumvent this problem. Perhaps the most satisfying way would be to have a monitor program that is aware of the complete state of the system, and that could determine whether or not the actions being taken are bringing the robot closer to the goal.

An alternative would be to have each ILA keep a record of whether or not its actions are leading toward the solution of its problem.

The simplest kind of record for an ILA to keep is a count of the number of times it has taken each action. In many cases, if an action has been taken once or twice before, and if

the predicates are calling for it to be taken again, then the ILA can assume that no progress is being made and return control to the calling program. This strategy can be improved by computing a limit on the number of allowed repetitions, and making this limit depend on the task. For example, if the action is to take the next step in a plan, the limit should obviously be related to the number of steps in the original plan. Both of these strategies can be criticized on the grounds that they are indirect and possibly very poor measures of the progress being made. However, they constitute a frequently effective, simple heuristic, and will be used in our initial implementation of the ILAs.

4. Status and Implementation

As mentioned earlier, none of the ILAs has been implemented to date. However, some 15 have been sufficiently well defined to allow coding to begin. These are listed in Table 4 together with the ILAs that they call. The specification of the ILAs has also led to the specification of a number of specialized planning and information-gathering routines. The planning routines include programs for planning pushing sequences, tours from room to room, and trips within a single room. These will be developed along the lines of the navigation routines that were one of our earliest efforts on this project. The information-gathering routines are primarily special-purpose programs for processing television pictures. For example, PICLOC is a special-purpose routine that uses landmarks to update the location of the robot, and CLEARPATH analyzes a picture to see whether or not the path to the goal is clear. These routines are described in Chapter Six and Appendix B.

One aspect of implementing the ILAs that has not yet been resolved concerns whether the ILAs should be written as ordinary LISP programs, or should be kept in tabular form as data for an interpreter. It is quite easy to go from a representation such as that in Table 3 to a LISP program realization; the basic structure is merely a COND within a PROG. However, the use of an interpreter would simplify the implementation of the loop suppressor, and would also simplify monitoring and the incorporation of diagnostic messages. In addition, the same program that interprets the ILAs might be used to interpret the plans produced by STRIPS; if we can make these structures identical, the same executive program will be usable for both. Uniformity in program structure is also important for the plan generalization ideas (to be discussed in Chapter Eight).*

**From [10], pages 25-32.*

INTERMEDIATE LEVEL ACTIONS. ROUTINES MARKED BY ASTERISKS ARE VIEWED AS PRI

ILA	Routines Called	Comment
PUSH3	PLANOBMOVE*, PUSH2	Can plan and execute a se
PUSH2	PICLOC*, OBLOC*, NAVTO, ROLLBUMP, PUSH1	Check if object being pus
PUSH1	ROLL*	Basic push routine; assum
GETTO	GOTOROOM, NAVTO	Highest level go-to routi
GOTOROOM	PLANTOUR*, GOTOADJROOM	Can plan and execute a se
GOTOADJROOM	DOORPIC*, ALIGN, NAVTO, BUMBLETHRU	Tailored for going throug
NAVTO	PLANJOURNEY*, GOTO1	Can plan and execute a tr
GOTO1	CLEARPATH*, PICDETECTOB*, GOTO	Recovers from errors due
GOTO	PICLOC*, POINT, ROLL2	Executes single straight-
POINT	PICTHETA*, TURN2	Orients robot toward goal
TURN2	TURNBACK*, TURN1	Responds to unexpected bu
TURN 1	TURN*	Basic turn routine; expec
ROLL2	ROLLBACK*, ROLL1	Responds to unexpected bu
ROLL1	ROLL*	Basic roll routine that e
ROLLBUMP	ROLLBACK*, ROLL1	Basic roll routine that e

Table 4: INTERMEDIATE LEVEL ACTIONS*

*From [10], page 91.

The second excerpt describes the ILAs as they were implemented:

A. Introduction

The Intermediate-Level Actions (ILAs) are the action routines associated with the STRIPS operators (see Chapter Seven). Here we distinguish "action routines" from "operators" on the following basis: operators are used for planning, and the corresponding action routines are invoked to actually move the robot. The ILAs are written in a language we call Markov because of its resemblance to Markov algorithms. There is a large body of auxiliary LISP functions that accompanies the ILAs, but we will confine the present discussion to a brief description of the Markov language and a brief exposition of the current ILAs and the intraroom navigation algorithm.

B. The Markov Language

The central part of the Markov language is the Markov table, specifying actions to be performed and the criteria for determining their sequence. The format of a Markov table is an ordered collection of rows of identical format. Each row starts with a label, which is followed by a predicate, a sequence of actions to be performed, and finally the label of some other line in the table. This last item (which we have been calling the "go-to") can optionally specify that execution of the table could cease, causing the calling routine's execution to resume in the conventional subroutine fashion. The characteristic execution pattern is a sequential scan through the table's rows, testing the predicates one by one until a row is found whose predicate is true. Then the scan terminates and the actions (if any) in that row are performed, and the go-to is followed; it will either indicate completion of the execution of the table, or it will name a line in the table at which the scan is to recommence. When the Markov table is first entered, the scan begins with the first line in the table. Execution may be terminated in three ways: it can be completed explicitly, by reaching a special go-to; the sequential scan can get to the bottom of the table without having found a line with a true predicate; and finally, an action can be fruitless, which will cause a loop suppressor to terminate execution of the table. In all three cases, there is only one form of return from a Markov table, and the calling routine

The *actions* called for in an ILA may be LLAs, other ILAs, or arbitrary programs (usually coded in LISP). Since the Markov interpreter is itself a LISP program, an ILA can call itself recursively.

The "go-to" part of a Markov table line is interpreted after completion of the action part. In its simplest case, the "go-to" consists of the label of a line at which to continue the search for a true predicate. If several lines have the given label, one of the lines is arbitrarily chosen; if no lines have the given label, one of the lines is arbitrarily chosen; if no lines have it or if it is NIL, execution is terminated. (NIL is our conventional explicit return.) The other case involves "loop suppression" and will be discussed below.

A Markov table is generally a sequence of actions that would transform an initial state into a final "goal" state via a linear sequence of intermediate states. Whether an action is applicable to a particular state can usually be tested by a relatively simple predicate—the one heading the table line with the action. Since actions in the real world frequently fail to achieve their desired results, the Markov interpreter determines which action to execute by testing the state predicates one by one, starting from the goal predicate (on the top line) and working backward (i.e., down the table) until a true predicate is found. Markov operators typically follow the execution of any component action by starting again with the goal predicate. In its simplest form, each line of a Markov table would contain one of the state predicates and the operator to be applied to that state; its "go-to" would specify the first line, which contained the goal predicate and an explicit return. Falling off the end of a Markov table thus corresponds either to a drastic failure of one of the component actions or to an inappropriate application of the Markov operator. Of course, persistent failure of a component action to achieve its desired effect, i.e., to produce a state satisfying a predicate higher in the table, would cause indefinite looping in such a Markov table. To circumvent this possibility without requiring specific consideration in each Markov table, we introduced "loop suppression" into the Markov interpreter. Whenever the predicate of a line is found to be true, a counter is incremented and checked against a limit before the line's action is executed; if the counter becomes greater than the limit, then interpretation of the table is terminated without execution of the action. Thus, if the limit for a line is three (this is the current default value) then the action(s) on that line will be executed a maximum of three times; if the line's predicate is found true a fourth time, the table will return to the operator that invoked it. Of course, one can specify a limit for a table line rather than accepting the default value. There is an

alternative form for the "go-to" just for this purpose: rather than being just a label, it can be a two-element list. In this case, the first element is the label, and the second element is the loop-suppression limit for that line; it is evaluated only once, at the time of the first loop-suppression check for that line.

Table 5 illustrates the Markov language by presenting the actual code for the lowest-level ILA that pushes an object. Here, line 10 does some initialization; the action [i.e., the (SETQ XYTARG ...)] is always performed because its predicate T is always true. Then line 20's predicate checks whether the pushing operation is finished by means of its (NEARENOUGH OB XYTARG TOL) predicate; if this is the case, then no actions (i.e., NIL) are performed, and control jumps to the label CLEANUP for some post-processing before exit. Line 25's predicate similarly determines whether the object's position is known closely enough to continue the pushing operation. (This may not be the case either initially or as the result of the object dropping off the pushbar during a push.) Line 30 causes the table to exit (via CLEANUP) if the object is past its target. Line 40's predicate is true if the robot has just pushed the object into a wall, and finally, line 50's predicate is true if the robot has proper contact with the object. Line 10 and the lines starting with the label CLEANUP are representative of a more usual programming language, with the normal execution being sequential. Lines 20 through 50, however, have the characteristic execution pattern of the ILAs: a loop testing for the main goal and various subgoals and error conditions and recycling after any action is performed. This particular ILA is designed to be especially simple because it is intended to be embedded in several more layers of ILA before STRIPS becomes concerned with their robustness. Even STRIPS-visible ILAs are called by PLANEX (see Chapter 8) from its execution tables, so it is perfectly acceptable for this lowest-level pushing operator to fail as readily as it does.

C. The Actions

The following are brief descriptions of the present ILAs. The control relations among the ILAs and between them and the rest of the system are shown in Figure 7.

ILAs that affect the state of the world have responsibility for making corresponding changes to Shakey's axiom model of the current world. Such changes are mentioned below wherever relevant; "$" will be used to denote unspecified or changing values in the model.

GOTHRUDR(DOOR FROMRM TORM) moves the robot from room FROMRN to room TORM via door DOOR. It assumes only that the robot is in FROMRM; it uses NAVTO to get to the door and BUMBLETHRU to go through it.

BLOCK(DX RX BX) pushes box BX within room RX to a position blocking door DX. This routine directly replaces the axiom (UNBLOCKED DX RX) by (BLOCKED DX RX BX) in the model.

UNBLOCK(DX RX BX) pushes box BX within room RX to a position in which it does not block door DX; it directly replaces the axiom (BLOCKED DX RX BX) by (UNBLOCKED DX RX). This routine prefers to push the box to the far side of the door (as viewed from the initial position of the robot), but it will also consider the other push.

GOTO2(X) moves the robot into the vicinity of X if X is a door; it directly updates the (NEXTTO ROBOT $) axiom. A contemplated extension of GOTO2 is to permit X to be an object.

PUSH1(DIST OB TOL) is the lowest-level push; as such, it maintains OB's position and deletes the (NEXTO OB $) and (NEXTTO $ OB) axioms from the model. It pushes OB forward by DIST feet (within TOL feet); it assumes that the front horizontal catwhisker is on when it is entered, and it exits under any of the following conditions:

(1) It is unnecessary to push OB forward, i.e.:

(a) OB is within TOL of the implied goal point; or

(b) OB is past the goal point in the current heading.

(2) The pushbar comes on hard.

(3) The front horizontal catwhisker is off.

In any of these cases, the robot backs up 2 feet in an attempt to free its catwhiskers for normal navigation. The last argument TOL is optional and is defaulted to 1 foot if not supplied.

ROLL2(DIST TOL) is the lowest-level free-floor roll; as such it deletes the (NEXTTO ROBOT $) axiom from the model. It moves the robot forward by DIST feet (within TOL feet); if it engages a front catwhisker it asserts the (JUSTBUMPED ROBOT T) axiom and

backs away in an attempt to free the catwhisker. TOL is an optional parameter defaulted to 1 foot if not supplied; DIST may be negative.

BUMBLETHRU(FROMRM DOOR TORM) moves the robot from room FROMRM to room TORM through door DOOR. It assumes that the robot is initially in FROMRM and in front of door. It heads for the corresponding position in TORM and uses the catwhiskers (if necessary) to help it negotiate the door. It updates the (INROOM ROBOT $) and (NEXTTO ROBOT $) axioms in the model, and it is the most basic door-negotiating routine in the system. It uses the vision routine CLEARPATH before entering an unknown room.

PUSH(OBJECT GOAL TOL) is the highest-level ILA for pushing a box. Its three arguments are the name of an object, the goal coordinates to be pushed to, and the allowable tolerance. The tolerance argument may be omitted, in which case its value defaults to 2.0 feet.

The only precondition for PUSH is that Shakey and the OBJECT are in the same room. The routine calls FINDPATH (described below) to plan a path to GOAL from the current object location. PUSH will fail if any of the following conditions are true:

(1) OBJECT is not in a pushable location.

(2) No path of width W [W=MAX(WIDTH(OBJECT),WIDTH(ROBOT))] can be found from the current position of OBJECT to GOAL.

(3) No path can be found from the current position of the robot to the "pushplace" of OBJECT, i.e., Shakey cannot get behind OBJECT.

PUSH2(OBJECT GOAL TOL) is a straight-line push, envoked by PUSH to move OBJECT along successive legs of the planned path. PUSH2 attends to updating the positions of ROBOT and OBJECT. If the uncertainties in position exceed TOL, PICLOC updates the position of ROBOT or OBLOC the position of OBJECT (PICLOC and OBLOC are described in Chapter Six.)

A PUSH2 is accomplished in three basic stages:

(1) The robot navigates to the "pushplace" of OBJECT.

(2) The robot rolls forward and makes contact with the object with a front catwhisker, by using ROLLBUMP.

(3) PUSH1 is called, which turns on the overrides and causes the robot to roll forward the required distance.

NAVTO(GOAL TOL) will maneuver the robot to within TOL feet of the point GOAL. Like the PUSH ILA, it uses FINDPATH to plan the journey to GOAL. NAVTO will fail if no path is found; if a path exists, it uses POINT AND GOTO1 for each leg of the journey.

POINT(THETA TOL) attempts to turn the robot to within TOL degrees of bearing THETA. If necessary, the vision routine PICTHETA (Chapter Six) will be used to determine the bearing of the robot. A catwhisker engaged during the turn will cause the robot to turn back to its original bearing and then attempt to locate the object with PICBUMPED (Chapter Six).

GOTO1(GOAL TOL) moves the robot forward in a straight line to within TOL feet of GOAL. It will use ROLL2 to actually move the robot, or it will use vision under the following conditions:

(1) If the robot's location is uncertain (>TOL), it will update its position using PICLOC.

(2) If moving in an unknown room, it will use CLEARPATH.

(3) If the result of CLEARPATH is BLOCKED, it will use PICDETECTOB (Chapter Six) to enter information about the obstacle in the model.

(4) If the robot unexpectedly engages a catwhisker while rolling, PICBUMPED will locate the object and update the model.

ROLLBUMP(DIST TOL OBJECT) moves the robot forward DIST feet to engage a front catwhisker on the object OBJECT. It updates the (NEXTTO ROBOT $) predicate(s) in the model. If an object is not encountered within TOL feet of DIST, ROLLBUMP fails.

D. The Pathfinding Algorithm

FINDPATH(ROB G JOURN) is the routine to plan an intraroom path from ROB to G. The arguments ROB and G are each a list of X, Y coordinate pairs. JOURN is the type of journey to be undertaken, either ROLL or PUSH. If JOURN is ROLL, the

Table 5: MARKOV TABLE FOR THE LOWEST-LEVEL PUSHING ILA*

*From [11], page 41.

MARKOV TABLE FOR THE LOWEST-LEVEL PUSHING ILA

```
(DEFPROP PUSH1 PUSH1 (*: MARKOVTABLE NIL))

(DEFPROP PUSH1
         ((10. T ((SETQ XYTARG (XYTARG (OBPOS OB) (MLVFIND (QUOTE (THETA ROBOT $))) D
          (20. (NEARENOUGH OB XYTARG TOL) NIL CLEANUP)
          (25. (NOT (NEARENOUGH OB (OBPOS OB) TOL)) NIL C1)
          (30. (GREATERP (ABS (ANGLEDIF (BEARINGTO XYTARG (OBPOS OB)) (MLVFIND (QUOTE
               NIL
               CLEANUP)
          (40. (MEMQ (QUOTE HC) (WHISKERS))
               ((SETQ DOSETPOS NIL) (SETPUSHOBPOS OB (PLUS RADFRONT 0.5)))
               CLEANUP)
          (50. (MEMQ (QUOTE FH) (WHISKERS))
               ((OVRID 1.) (ROLL
                            (DIFFERENCE (DISTANCE XYTARG (OBPOS (QUOTE ROBOT)))
                                        (PLUS RADFRONT (MLVFIND (LIST (QUOTE RADIUS) (
                           (OVRID 0.)
                           (SETQ DOSETPOS NIL)
                           (SETPUSHOBPOS OB RADFRONT)
                           (MLDELETE (LIST (QUOTE NEXTTO) OB (QUOTE $)))
                           (MLDELETE (LIST (QUOTE NEXTTO) (QUOTE $) OB)))
               20.)
          (CLEANUP DOSETPOS ((SETPUSHOBPOS OB (PLUS RADFRONT 0.5))) C1)
          (C1 (FCWON) ((ROLLBACK) (ROLL -1.)) C2)
          (C2 T ((MLDELETE (QUOTE (NEXTTO ROBOT $)))) R))
         (*: MARKOVTABLE TABLE))

(DEFPROP PUSH1 (DIST OB TOL) (*: MARKOVTABLE PARAMETERS))

(DEFPROP PUSH1 ((TOL 1.) XYTARG (RADFRONT 1.5) (DOSETPOS T)) (*: MARKOVTABLE LOCALS))
```

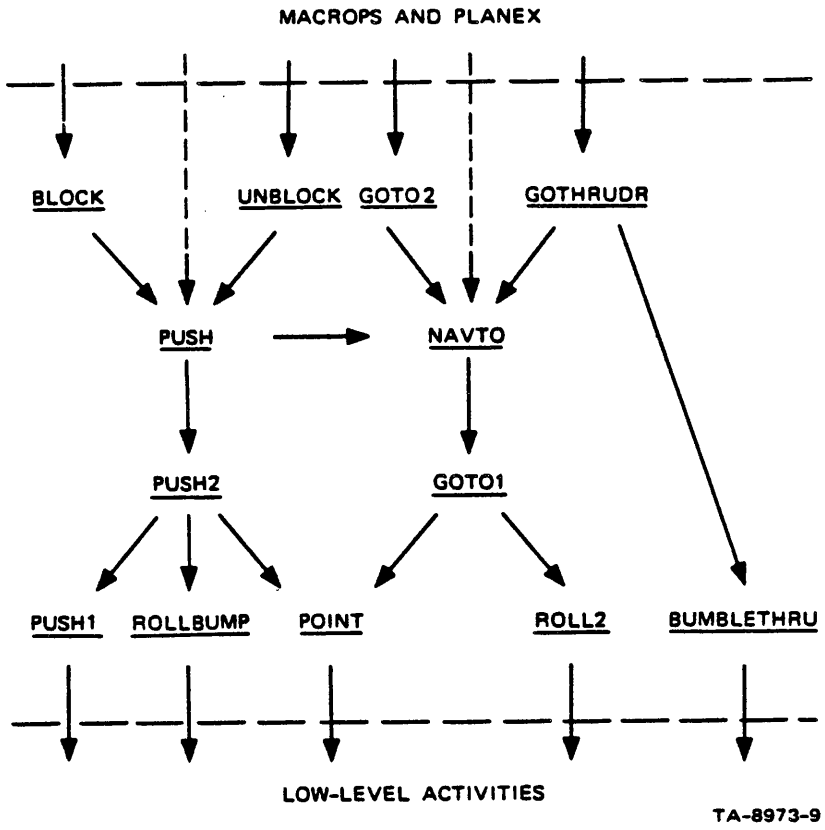


Figure 7: CONTROL STRUCTURE OF THE INTERMEDIATE LEVEL

function returns a path along which the robot can navigate from ROB to G. If JOURN is PUSH, the returned value is a path by which the robot can move a box at ROB to point G. In this case global variables PUSHOBNAME (name of the box) and OBRAD (radius of

The pathfinding algorithm is a breadth-first search of the tree of predecessors to G. At each node of the tree, FINDPATH tests for a direct-line path between ROB and the current node, say PN. If it exists, the path from PN to G is returned. Otherwise, the tree is grown one level deeper from PN by computing predecessors to that point. If no predecessors exist, the path from PN to G is removed from the tree, thus reducing the search space.

The predecessors to node PN are defined as the intersections of the tangent lines from ON and ROB around the first obstructing object in the straightline path connecting them. Thus, each point has at most two predecessors. Figure 8 illustrates one possible configuration that would generate the tree in Figure 9.

Before a computed predecessor is added to the tree, it is tested to determine whether it is within the room or within the region of another obstacle. It either condition is true (as for PØ in Figure 8), a shorter path (P5 P4) is computed using the tangents that generated PØ. If either of these points is unacceptable under the criterion just described, the entire search in that direction is abandoned, and the next node (in this case P3) is considered. A predecessor that is acceptable under this criterion is added to the tree *if* a straightline path exists between it and its parent node. Otherwise, predecessors are sought recursively to find a path from the parent node to its computed predecessor.

The searching in FINDPATH terminates, then, when either a path has been found or when the search tree is reduced to NIL. Thus, the path that is chosen (assuming at least one exists) is the first one found, that is, the one with the smallest number of legs in the journey. This criterion was chosen over a minimum-distance criterion to reduce the amount of subsequent thinking and execution time for the robot.*

**From [11], pages 37-49.*

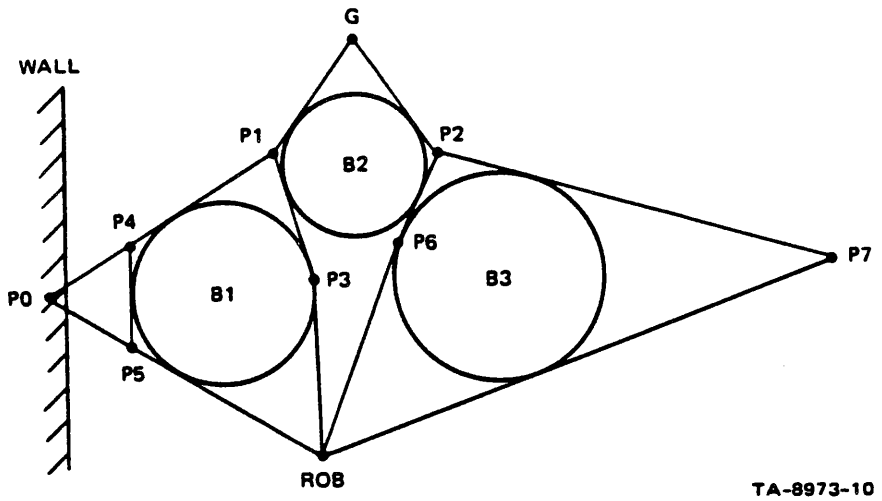


Figure 8: AN OBSTACLE CONFIGURATION FOR FINDPATH*

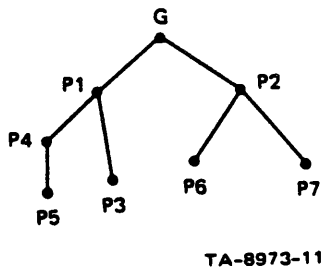


Figure 9: SEARCH TREE FOR CONFIGURATION OF FIGURE 8*

*From [11], page 48.

CHAPTER SIX

Vision Routines

We first present an overview of the main vision routines from [11].

A. Introduction

The current robot executive program never calls for a general visual scene analysis. Instead, under appropriate circumstances various of the intermediate-level actions (ILAs) call specific vision routines to answer certain specific questions. These specialized vision programs perform three basic tasks: locating and orienting the robot, detecting the presence of objects, and locating objects.

A summary of the six vision routines currently used by the ILAs is given below in Section C. PICLOC is described in Appendix B, and CLEARPATH is described briefly later. Most of the other routines make use of LOBLOC, which uses vision to locate accurately an object whose position is only roughly known.

The following section describes the operation of this routine in some detail.

B. Object Location

Given the approximate floor location of an object, LOBLOC takes a television picture of the object, analyzes the picture to find the exact coordinates, and enters this information in the robot's world model. This specialized task can be done more rapidly and with less chance for error by a special program than by performing a complete scene analysis and then extracting the desired answer from the resulting description. However, certain preconditions must be satisfied for LOBLOC to function properly. These are as follows:

(1) The approximate location must be sufficiently accurate and the object must be sufficiently small and unoccluded that at least two, and preferably three, lower corners of the object are in view.

(2) The object and the robot must be in the same room.

(3) The location of the robot with respect to the walls must be known to within approximately one foot.

The first action that LOBLOC performs is to pan and tilt the television camera so that the nominal floor position image is in the center of the picture. The resulting picture is taken at 60-line resolution to speed subsequent region analysis operations. However, before region analysis is begun, the program accesses the model to compute the image of the wall-floor boundary. Everything in the picture above this boundary is erased, thereby eliminating baseboards, door jambs, and other possible sources of confusion.

The resulting picture is then subjected to region analysis. That is, it is partitioned into elementary regions, and these regions are merged using the phagocyte and weakness heuristics [16]. The following regions are automatically deleted from the resulting region list:

(1) The region above the wall-floor boundary.

(2) All regions smaller than some threshold θ. (Currently $\theta = 4$ cells.)

The next major step is to identify the floor region. This is done by scoring each region. The features or properties that enter into this score are the area A, the ratio R of perimeter-squared to area, the average brightness B, and the lowest coordinate Z of the external contour. Letting A_{max} be the largest area, R_{max} the largest ratio, B_{max} the highest brightness, and Z_{min} the smallest coordinate, we compute the scoring function by

$$D^2 = \left(1 - \frac{A}{A_{max}}\right)^2 + \left(1 - \frac{R}{R_{max}}\right)^2 + \left(1 - \frac{B}{B_{max}}\right)^2 + \left(\frac{Z - Z_{min}}{60}\right)^2 .$$

The region for which D^2 is minimum is declared to be the floor.

The next major step is to inspect the n neighbors of the floor to find the ones that are most likely to be the faces of the object in question. Special tests are made to treat the simple cases where n happens to be 0, 1, or 2. In general, for each region neighboring the

floor we compute its area A and a quantity X which is a simple measure of the horizontal displacement of the region from the center of the picture. These features are combined in a scoring function:

$$D^2 = \left(1 - \frac{A}{A_{max}}\right)^2 + \left(\frac{X - X_{min}}{60}\right)^2 ,$$

and the region for which D^2 is minimum is declared to be one face of the object. The same criterion is used to select the other visible face from the neighbors of both the floor and the first face.

The major problem remaining is to identify the vertices where the corners of the object meet the floor. This is done by processing the common boundary between the face regions and the floor region. After simple straight-line connections are made between endpoints of any gaps, this common boundary consists of a chain of points along the lower edge of the object. The lowest point on this chain is taken to be the central vertex, and the corners on either side are found by the method of iterative end-point fits [17]. Once these three image points are determined, the support hypothesis is used to locate the corresponding points on the floor. The resulting coordinates can then be entered in the model under the name of a new object if the status of the room is unknown, or under the name of the nearest object if the status is known.

C. ILA Vision Routines

The following is a summary of the intermediate-level routines related to Shakey's visual system:

CLEARPATH (X Y) decides whether the path from (AT ROBOT $*) to (X Y) is clear. In analyzing pictures, it inspects only the image of the path to be traversed, and it uses the range finder to detect large, close objects. The value returned is either CLEAR, UNKNOWN, or (BLOCKED XO YO), where (XO YO) roughly locates an obstacle.

OBLOC (OB) uses the model information about the location of object OB and the routine LOBLOC to update (AT OB $*) and (DAT OB $*).

PICBUMPED (X Y) is called when a bump occurs at (X Y). If Shakey is in a room of known status, PICBUMPED calls PICLOC; otherwise it calls PICDETECTOB (X Y).

PICDETECTOB (X Y) uses LOBLOC to locate the object near (X Y). If Shakey is in a room of known status, and if OB is the nearest object, (AT OB $*) and (DAT OB$*) are updated; otherwise a new object is entered in the model.

PICLOC uses the landmark routine (Appendix B) to update (AT ROBOT $*), (DAT ROBOT $*), (THETA ROBOT $), and (DTHETA ROBOT $).

PICTHETA updates (THETA ROBOT $) and (DTHETA ROBOT $). Intended to be used before a long, straight-line journey, PICTHETA currently calls PICLOC.*

Additional material about Shakey's vision system was reported in [10].

Vision Programs for Intermediate-Level Actions

The special-purpose vision programs basically perform only three functions: orienting and locating the robot, detecting the presence of objects, and locating objects. We shall consider each of these functions in turn.

When the environment of the robot is represented accurately and completely in the model, the chief role of vision is to provide feedback to update the robot's position and orientation. Angular orientation information is often needed in advance of a relatively long trip down a corridor, where a small angular error might be significant. The simplest way to obtain orientation feedback is to find the floor/wall boundary in the picture, project it onto the floor, and compare this result with the known wall location in the model; any observed angular discrepancy can be used to correct the stored value of the robot's orientation.

For maneuvers such as going through a doorway, both the robot's position and orientation must be accurately known. This information can be obtained from a picture of a known

---

**From [11], pages 51-54.*

point and line on the floor. Such distinguished points and lines are called landmarks, and include doorways, concave corners, and convex corners. The basic program for finding such landmarks is described in Appendix B. The program has undergone several refinements and improvements, and now works with the model described in Chapter Three. Execution time is essentially the time required to pan, tilt, and turn on the camera.* Concurrently, the accuracy is limited by mechanical factors to between 5 and 10 percent in range and 5 degrees in angle. Increased accuracy, if needed, can be obtained by improving the pan and tilt mechanism for the camera.

Before the robot starts a straight-line journey, it may be desirable to check that the path is indeed clear. A simple way to do this is to find the image of the path in the picture and examine that trapezoidal-shaped region for changes in brightness that might indicate the presence of an obstructing object. This is a simple visual task, and a program implementing it has been written. In its current form the program uses the Roberts-cross operator to detect brightness changes. When we first ran the program, we were surprised to discover that at steep camera angles the texture in the tile floor can be detected and give rise to false alarms. This is an instance of a major shortcoming of special-purpose vision routines, namely, the failure of simple criteria to cope with the variety of circumstances that can arise. This particular problem can be solved by requiring a certain minimum run-length of gradient. However, shadows and reflections can still cause false alarms, and the only solution to some of these problems is to do more thorough scene analysis.**

---

*Since the camera, television control unit, and television transmitter draw a large amount of power from the batteries, they are normally off. Approximately ten seconds is required from the time these units are turned on to the time that a picture can be taken.

**_From [10], pages 41-43._

CHAPTER SEVEN

STRIPS

Shakey used a planning system called STRIPS (an acronym based on STanford Research Institute Problem Solver) to chain together ILAs that would accomplish specific goals. STRIPS was one of the important early problem-solving systems. The original version of this program is described in detail in a paper [18]; a somewhat modified story appears in [19]. More recent hierarchical planning systems, such as NOAH [20] and SIPE [21], would now be more appropriate than STRIPS for robot planning. The following excerpt is a summary of STRIPS that appeared in a paper and an SRI AI Center Technical Note [22] about learning and executing plans.

Description

Because STRIPS is basic to our discussion, let us briefly outline its operation. The primitive actions available to the robot vehicle are precoded in a set of action routines. For example, execution of the routine GOTHRU(D1,R1,R2) causes the robot vehicle actually to go through the doorway, D1, from room R1 to room R2. The robot system keeps track of where the robot vehicle is and stores its other knowledge of the world in a *model* composed of well-formed formulas (wffs) in the predicate calculus. Thus, the system knows that there is a doorway D1 between rooms R1 and R2 by the presence of the wff CONNECTSROOMS(D1,R2,R2) in the model.

Tasks are given to the system in the form of predicate calculus wffs. To direct the robot to go to room R2, we pose for it the goal wff INROOM(ROBOT,R2). The planning system, STRIPS, then attempts to find a sequence of primitive actions that would change the world in such a way that the goal wff is true in the correspondingly changed model. In order to generate a plan of actions, STRIPS needs to know about the effects of these actions; that is, STRIPS must have a model of each action. The model actions are called *operators* and, just as the actions change the world, the operators transform one model

into another. By applying a sequence of operators to the initial world model, STRIPS can produce a sequence of models (representing hypothetical worlds) ultimately ending in a model in which the goal wff is true. Presumably the, execution of the sequence of actions corresponding to these operators would change the world to accomplish the task.

Each STRIPS operator must be described in some convenient way. We characterize each operator in the repertoire by three entities: an *add function*, a *delete function*, and a *precondition wff*. The meanings of these entities are straightforward. An operator is applicable to a given model only if its precondition wff is satisfied in that model. The effect of applying an (assumed applicable) operator to a given model is to delete from the model all those clauses specified by the delete function and to add to the model all those clauses specified by the add function. Hence, the add and delete functions prescribe how an operator transforms one state into another; the add and delete functions are defined simply by lists of clauses that should be added and deleted.

Within this basic framework STRIPS operates in a GPS-like manner [23]. First, it tries to establish that a goal wff is satisfied by a model. (STRIPS uses the QA3 resolution-based theorem prover [15] in its attempts to prove goal wffs.) If the goal wff cannot be proved, STRIPS selects a "relevant" operator that is likely to produce a model in which the goal wff is "more nearly" satisfied. In order to apply a selected operator, the precondition wff of that operator must of course be satisfied: This precondition becomes a new subgoal and the process is repeated. At some point we expect to find that the precondition of a relevant operator is already satisfied in the current model. When this happens the operator is *applied*; the initial model is transformed on the basis of the add and delete functions of the operator, and the model thus created is treated in effect as a new initial model of the world.

To complete our review of STRIPS we must indicate how relevant operators are selected. An operator is needed only if a subgoal cannot be proved from the wffs defining a model. In this case the operators are scanned to find one whose effects would allow the proof attempt to continue. Specifically, STRIPS searches for an operator whose add function specifies clauses that would allow the proof to be successfully continued (if not completed). When an add function is found whose clauses do in fact permit an adequate continuation of the proof, then the associated operator is declared relevant; moreover, the substitutions used in the proof continuation serve to instantiate at least partially the arguments of the operator. Typically, more than one relevant operator instance will be found. Thus, the

entire STRIPS planning process takes the form of a tree search so that the consequences of considering different relevant operators can be explored. In summary, the "inner loop" of STRIPS works as follows:

(1) Select a subgoal and try to establish that it is true in the appropriate model. If it is, go to Step 4. Otherwise,

(2) Choose as a relevant operator one whose add function specifies clauses that allow the incomplete proof of Step 1 to be continued.

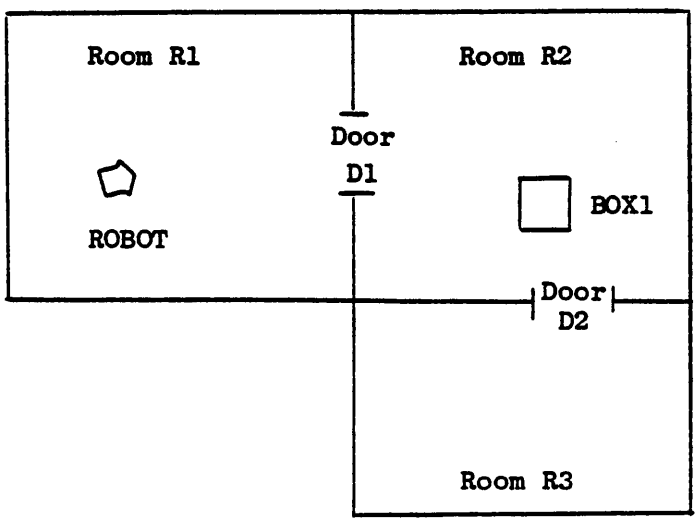
(3) The appropriately instantiated precondition wff of the selected operator constitutes a new subgoal. Go to Step 1.

(4) If the subgoal is the main goal, terminate. Otherwise, create a new model by applying the operator whose precondition is the subgoal just established. Go to Step 1.

The final output of STRIPS, then, is a list of instantiated operators whose corresponding actions will achieve the goal.

An Example

An understanding of STRIPS is greatly aided by an elementary example. The following example considers the simple task of fetching a box from an adjacent room. Let us suppose that the initial state of the world is as shown below:



Initial Model

Mo: INROOM(ROBOT,R1)
CONNECTS(D1,R1,R2)
CONNECTS(D2,R2,R3)
BOX(BOX1)
INROOM(BOX1,R2)

.
.
.

$(\forall x\ \forall y\ \forall z)[\text{CONNECTS}(x,y,z) \Rightarrow \text{CONNECTS}\ (x,z,y)]$

Goal wff

Go: $(\exists x)\ [\text{BOX}(x) \wedge \text{INROOM}(x,R1)]$

We assume for this example that models can be transformed by two operators GOTHRU and PUSHTHRU, having the descriptions given below. Each description specifies an *operator schema* indexed by schema variables. We will call schema variables *parameters*, and denote them by strings beginning with lower-case letters. A particular member of an operator schema is obtained by instantiating all the parameters in its description to constants. It is a straightforward matter to modify a resolution theorem prover to handle wffs containing parameters [18], but for present purposes we need only know that the modification ensures that each parameter can be bound only to one constant; hence, the operator arguments (which may be parameters) can assume unique values. (In all of the following we denote constants by strings beginning with capital letters and quantified variables by x, y, or z):

GOTHRU(d,r1,r2)

(Robot goes through Door d from Room r1 into Room r2.

Precondition wff

$\text{INROOM}(\text{ROBOT},r1) \wedge \text{CONNECTS}(d,r1,r2)$

Delete List

INROOM(ROBOT,$)

Our convention here is to delete any clause containing a predicate of the form INROOM(ROBOT,$) for any value of $.

Add List

INROOM(ROBOT,r2)

PUSHTHRU(b,d,r1,r2)

(Robot pushes Object b through Door d from Room r1 into Room r2.)

Precondition wff

INROOM(b,r1) ∧ INROOM(ROBOT,r1) ∧ CONNECTS(d,r1,r2)

Delete List

INROOM(ROBOT,$)

INROOM(B,$)

Add List

INROOM(ROBOT,r2)

INROOM(b,r2).

When STRIPS is given the problem it first attempts to prove the goal G_0 from the initial model M_0. This proof cannot be completed; however, were the model to contain other clauses, such as INROOM(BOX1,R1), the proof attempt could continue. STRIPS

determines that the operator PUSHTHRU can provide the desired clause; in particular, the partial instance PUSHTHRU(BOX1,d,r1,R1) provides the wff INROOM(BOX1,R1).

The precondition G_1 for this instance of PUSHTHRU is

G_1: INROOM(BOX1,r1)
$\wedge$ INROOM(ROBOT,r1)
$\wedge$ CONNECTS(d,r1,R1).

This precondition is set up as a subgoal and STRIPS tries to prove it from M_0.

Although no proof for G_1 can be found, STRIPS determines that if r1 = R2 and d = D1, then the proof of G_1 could continue were the model to contain INROOM(ROBOT,R2). Again STRIPS checks operators for one whose effects could continue the proof and settles on the instance GOTHRU(d,r1,R2). Its precondition is the next subgoal, namely:

G_2: INROOM(ROBOT,r1)
$\wedge$ CONNECTS(d,r1,R2).

STRIPS is able to prove G_2 from M_0, using the substitutions r1 = R1 and d= D1. It therefore applies GOTHRU(D1,R1,R2) to M_0 to yield:

M_1: INROOM(ROBOT,R2)
CONNECTS(D1,R,R2)
CONNECTS(D2,R2,R3)
BOX(BOX1)
INROOM(BOX1,R2)
.
.
.
$(\forall x\ \forall y\ \forall z)$[CONNECTS(x,y,z) $\Rightarrow$ CONNECTS(x,z,y)].

Now STRIPS attempts to prove the subgoal G_1 from the new model M_1. The proof is successful with the instantiations r1 = R2, d = D1. These substitutions yield the operator instance PUSHTHRU(BOX1,D1,R2,R1), which applied to M_1 yields

M_2: INROOM(ROBOT,R1)
CONNECTS(D1,R1,R2)
CONNECTS(D1,R2,R3)
BOX(BOX1)
INROOM(BOX1,R1)

.
.
.

(∀x ∀y ∀z) [CONNECTS(x,z,y)].

Next, STRIPS attempts to prove the original goal, G_0, from M_2. This attempt is successful and the final operator sequence is

GOTHRU(D1,R1,R2)
PUSHTHRU(BOX1,D1,R2,R1).*

From [22], pages 4-11 of Technical Note.

CHAPTER EIGHT

LEARNING AND EXECUTING PLANS

Once a plan to accomplish a goal has been constructed, the robot executive system, called PLANEX, executes it. If problems arise during execution, PLANEX must also decide how to modify the plan it is executing or whether to construct a new plan. The Shakey system also was able to learn generalized versions of the plans it constructed that could be used to help accomplish subsequent tasks. These capabilities were described in a paper [22] and summarized in one of the Shakey technical reports [11]. The following excerpt is from that report:

A. Introduction

The basic problem-solving system used by Shakey is STRIPS, a system that makes use of a combination of heuristic search and formal deductive techniques. However, STRIPS in its original form is limited to constructing a plan for solving a specific problem. In this section we describe new:

(1) Procedures for constructing "generalized" plans that are applicable to a large family of problems (in addition to the specific problem that motivated the planning process).

(2) Methods for storing, selecting, and monitoring the use of generalized plans while a task is actually being carried out.

The recently developed methods for storing and using generalized plans allow us:

(1) To store a generalized plan as a sequence of, say, n parameterized operators.

(2) To use as a single operator in a subsequent planning process many of the legal subsequences among the 2^n - 1 subsequences of the original sequence of n operators.

(3) To identify for monitoring purposes exactly those effects of a selected subsequence that are necessary for the success of the new plan.

As a rough illustration of the use of these capabilities, suppose that we already have a generalized plan for closing a door and turning off a light. We are now given the task of just turning off some particular light. The methods to be described will extract from the original plan the appropriate subsequence of operators needed to turn off the light. Suppose now that the subsequence of operators, or *subplan*, for turning off the light also has the effect of leaving the robot pointing in a specified direction. If this effect is a legitimate *side-effect*—that is, if the successful execution of the plan does not require the robot to be pointing in a specified direction—then the methods described will identify this fact and the final robot orientation will not be monitored during plan execution. Hence, the plan execution mechanism will not reject as "unsuccessful" an execution that has failed only in a detail irrelevant to the task at hand.

The processes for storing a generalized plan begin with the creation by STRIPS of a generalized plan, or *macro operator*—that is, a sequence of n operators whose arguments are parameters. During the creation of this plan, STRIPS performed proofs demonstrating that each operator was in fact applicable at the time it was used. We assume throughout this section the availability of both the STRIPS plan and certain information about the structure of the proofs performed by STRIPS to generate the plan. We also assume the availability of *descriptions* of each operator used in the plan. An operator description consists of three things: a *precondition formula*, which must be provable from a model if the operator is to be applied to that model; an *add-list*, specifying clauses added to the model; and a *delete function* (represented as a list of literals), which maps a set of clauses into a subset of itself that remains true after the operator has been applied.

B. Storage of a Generalized Plan

We store a generalized plan in the the form of a triangular table* as shown in figure 10. The columns of the table, with the exception of column 0, are labeled with the names of the operators of the plan, in this example OP_1, ...,OP_4. For each column i, i = 1, ...,4, we place in the top cell the add-list A_i of operator OP_i. Going down the i^{th} column, we place

The late John Munson of the SRI Artificial Intelligence Center originally suggested this tabular format.

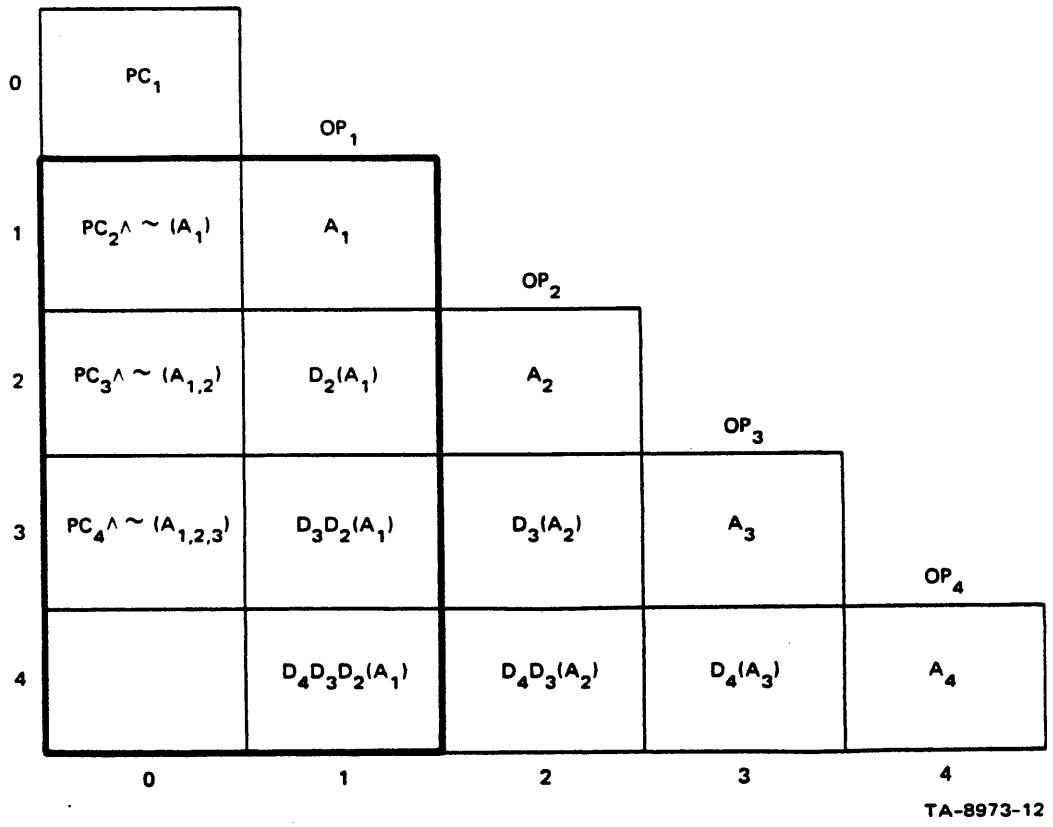


Figure 10: TYPICAL MACROP

in consecutive cells the portion of A_i that survives the application of subsequent operators. This is indicated by the delete function D_i, i = 2, 3, 4, that maps an add-list into the subset of itself remaining true after the application of OP_i. (The delete function D_1 of OP_1 is applied to the model in which MACROP is applied, and not to any of the add-lists.) Thus, cell (2,1) contains $D_2(A_1)$, which is the portion of A_1 still true after OP_2 is applied. Cell (3,1) contains $D_3(D_2(A_1))$ =$D_3D_2(A_1)$, which is the subset of A_1 that survives the application of both OP_2 and OP_3.

We can now interpret the content of the i^{th} row of the table, excluding the first column. Since each cell in the i^{th} row (excluding the first) contains statements added by one of the first i operators and not deleted by any of the first i operators, we see that the union of the cells in the i^{th} row (excluding the first cell) specified the add-list obtained by applying in sequence OP_1, ...,OP_i. We denote by $A_{1, ...,i}$ the add-list achieved by the first i operators applied in sequence. The union of the cells in the bottom row of a triangle table specified the add-list of the complete macro operator.

Let us now consider the first column of the triangle table, which we have so far ignored. Loosely, the statements in row i of column zero are involved with the precondition formula PC_{i+1} of OP_{i+1}. To be more specific, cell (i,0) contains clauses needed to prove PC_{i+1} but not contained in $A_{1, ...,i}$. We will call the set of clauses (axioms) used to prove a formula the *support* of that formula. The clauses in cell (i,0) are therefore the portion of the support of PC_{i+1} that was true in the initial state. (In Figure 10, we have used the notation $PC_i \wedge \sim A_{1, ...,i}$ to indicate the contents of cell (i,0).) The remaining part of the support of PC_i is supplied by applying in sequence OP_1, ...,OP_i. The i^{th} row of the table, then, contains the complete support of the precondition of OP_{i+1}. It is convenient to flag the clauses in row i that are the support of PC_{i+1}, and hereafter speak of marked clauses; by construction, obviously, all clauses in column zero are marked.

C. Planning with Generalized Plans

1. General Approach

In the preceding section, we described the construction of triangle tables for storing generalized plans. Now let us consider how a generalized plan will be used by STRIPS during a subsequent planning process.

The first thing to emphasize is that the i^{th} row of a triangle table (excluding its first cell) represents the add-list $A_{1,...,i}$; an n-row table presents STRIPS with n alternative add-lists, any one of which can be used to reduce a difference encountered by STRIPS during its normal planning process. STRIPS selects a particular add-list in the usual fashion by testing the relevance of that add-list with respect to the difference currently being considered. Suppose for a moment that STRIPS selects the i^{th} add-list $A_{1,...,i}$, $i < n$. Since this add-list is achieved by applying in sequence OP_1 ,...,OP_i, we will obviously not be interested in the application of OP_{i+1} ,...,OP_n, and will therefore not be interested in establishing any of the preconditions PC_{i+1},...,PC_n. Now in general, some steps of a plan are needed only to establish preconditions for subsequent steps. If we lose interest in the *tail* of a plan—that is, in the last (n - i) operators—then we may be able to achieve some economies by omitting those operators among the first i whose sole purpose is to establish preconditions for the tail. Conceptually, then, we can think of a single triangle table as representing a family of generalized operators. Upon the selection by STRIPS of a relevant add-list, we must extract from this family an economical parameterized operator achieving the add-list. STRIPS must then be provided with a complete

description—precondition wff, add-list, and delete function—of the extracted operator so that it can be used during the planning process.

In the following paragraphs, we will explain by means of an example an algorithm for accomplishing this task of *operator extraction*.

2. The Operator Extraction Algorithm

Consider the illustrative triangle table shown in Figure 11. Each of the numbers within cells represents a single clause. The circled clauses are "marked" in the sense described earlier; that is, they are used to prove the precondition of the operator whose name appears on the same row. A summary of the structure of this plan is shown below, where "I" refers to the initial state and "F" to the final state:

Operator	Precondition Support Supplied By	Precondition Support Supplied To
OP_1	I	OP_4
OP_2	I	OP_5
OP_3	I	OP_7, F
OP_4	I,OP_1	F
OP_5	I,OP_2	OP_6, F
OP_6	I,OP_5	OP_7
OP_7	I,OP_3,OP_6	F

Suppose now that STRIPS selects $A_{1,\ldots,6}$ as the desired add-list and, in particular, selects clause 16 and clause 25 as the particular members of the add-list that are relevant to reducing the difference of immediate interest. These clauses have been marked on the table with a dot. The operator extraction algorithm proceeds by examining the table to determine what effects of individual operators are not needed to produce clauses 16 and 25. First, OP_7 is obviously not needed; we can therefore remove all circle marks from row 6, since those marks indicate the support of PC_7. We now inspect the columns, beginning with column 6 and going from right to left, to find the first column with no marks of either kind. Column 4 is the first such column. The absence of marked clauses in column

4 means that the clauses added by OP_4 are not needed to reduce the difference and are not required to prove the precondition of any subsequent operator; hence we delete OP_4 from the plan and unmark all clauses in row 3. Continuing our right-to-left scan of the columns, we note that column 3 contains no marked clauses. (Recall that we have already unmarked clause 18.) We therefore delete OP_3 from the plan and unmark all clauses in row 2. Continuing the scan, we note that column 1 contains no marked entries (we have already unmarked clause 11), and therefore delete OP_1 and the marked entries in row 0.

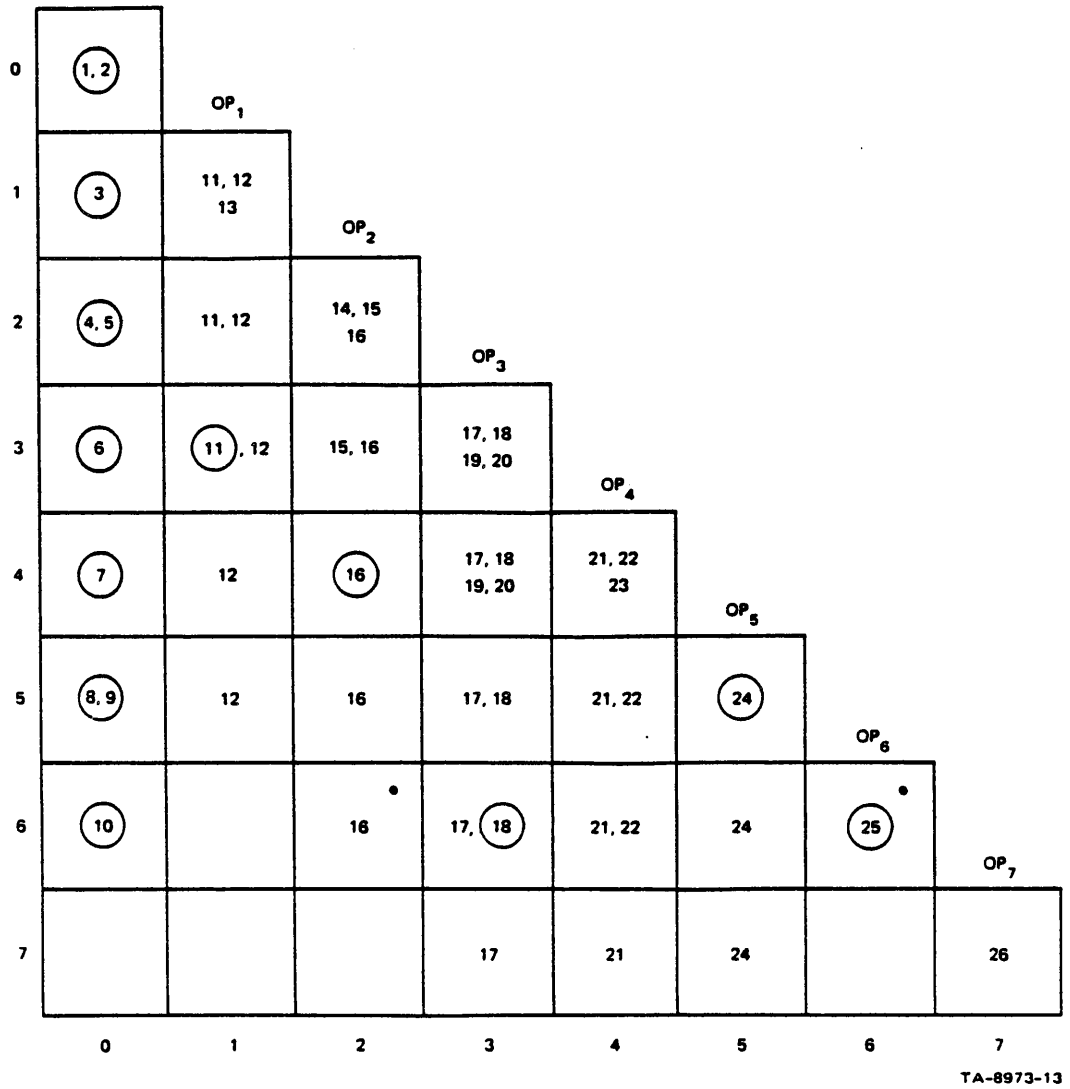


Figure 11: MACROP WITH MARKED CLAUSES

The result of the table-editing process just described is shown in Figure 12. (The question mark in cell (2,1) will be explained momentarily.) A summary of the structure of this plan is shown below:

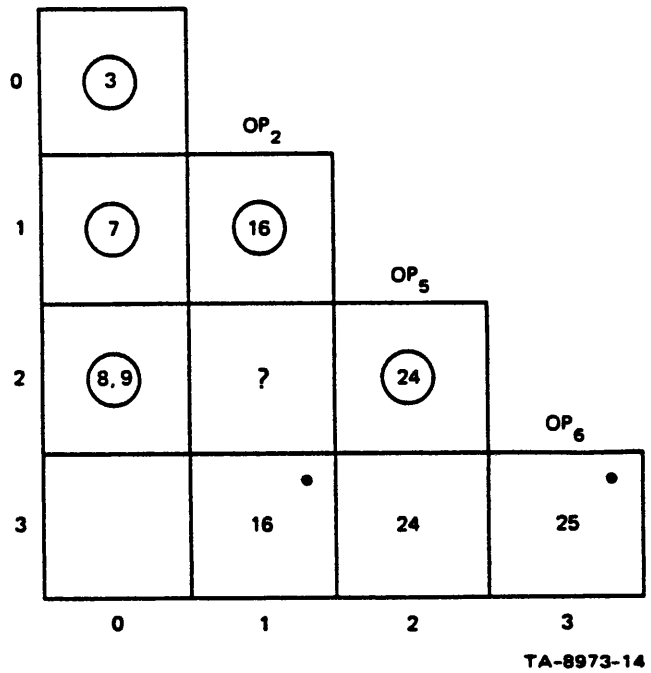


Figure 12: MACROP AFTER EDITING

Operator	Precondition Support Supplied By	Precondition Support Supplied To
OP_2	I	OP_5,F
OP_5	I,OP_2	OP_6
OP_6	I,OP_5	F

We have thus reduced the seven-step generalized plan we started with to a compact three-step plan that specifically produces an add-list containing the relevant clauses.

Now that an operator achieving a desired add-list has been extracted, we must provide STRIPS with its description. The precondition wff is obvious; it consists of the

conjunction of all clauses in column 0. The computation of the add-list and delete function of the new operator is a little more complicated. First, notice in Figure 11 that clauses 14, 15, and 16 are added by OP_2. Clause 14 is evidently deleted by OP_3 since it does not appear in cell (3.2). The extracted plan, however, does not include OP_3, and we cannot tell whether clause 14 would survive the application of OP_5 or OP_6 in the extracted plan—hence the question mark in Figure 12. Furthermore, cell (3,1) may contain more clauses than shown. This example illustrates the necessity of computing a new add-list and delete function for the extracted operator.

The computation of a new add-list and delete function for a macro operator is based on the add-lists and delete functions of the component operators. Suppose the macro operator of Figure 12 is applied to some state S_i (in which we assume that clauses 3, 7, 8, and 9 are true). Since STRIPS does deletions before additions, we can write the resulting state S_f as:

$$S_f = D_6(D_5(D_2(S_i) + A_2) + A_5) + A_6 \quad ,$$

where we have used "+" to mean set union. Now it is not difficult to show that delete functions distribute over set union, that is, to show for any set A and B and any delete function D that

$$D(A + B) = D(A) + D(B)$$

Hence, we can write the final state S_f as:

$$S_f = D_6D_5D_2(S_i) + D_6D_5(A_2) + D_6(A_5) + A_6 \quad .$$

Since this has the form $S_f = D(S_i) + A$, we see that the delete function of the macro operator is the composed function

$$D_6D_5D_2 \quad ,$$

and that its add-list is

$$D_6D_5(A_2) + D_6(A_5) + A_6 \quad .$$

It is interesting to note that this add-list is precisely the last row of the triangle table constructed as described in the previous section, the plan OP_2. OP_5, OP_6. In general, we can say that the add-list of a macro operator is given by the last row of its triangle table representation, and that its delete function is given by the composition of the component delete functions.

3. Refinements

In the previous paragraphs, we outlined an algorithm for extracting from a generalized plan a subsequence of operators that add particular clauses to a model. We would now like to describe two refinements: one needed to avoid certain inconsistencies that could otherwise occur, and one for achieving further economies when more than one level of triangle tables are involved.

a. Add-List Refinement

Consider a simple generalized plan consisting of two consecutive PUSH operators, each of which pushes a (parameterized) object to a (parameterized) place. The triangle table for this plan might be as shown in Figure 13 where for simplicity we have assumed that the PUSH operator has no precondition and hence column 0 is empty. Because the clause AT(OB1,P1) appears in cell (2,1), we know that this clause was not deleted by the second push operator. Suppose now that STRIPS selects row 2 as an add-list. By instantiating OB1 and OB2 to the same object name, and instantiating P1 and P2 to distinct locations, we evidently have a plan for achieving a state in which the same object is simultaneously at two different places! The source of this embarrassment lies in the delete mechanism used by STRIPS, which we now examine in some detail.

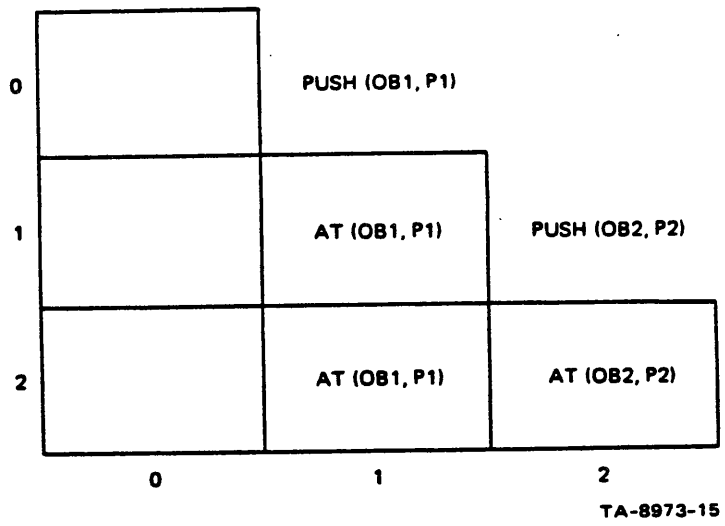


Figure 13: GENERALIZED PLAN FOR TWO-PUSH MACROP

The delete function of an arbitrary STRIPS operator is specified by a *delete-list* consisting of a set of literals. If the operator is applied to a state S, then STRIPS deletes from S every clause containing a literal unifying (without regard to sign) with any member of the delete-list. If a potential unification involves parameters, as it often does, then the unification can be made only if it does not contradict any existing bindings of the parameters to constants. To continue our example, suppose the second push operator is applied to the parameterized state S:

AT(OB1, P1)
AT(OB2, P3).

The delete-list of the second push operator, we assume, contains the single literal AT(OB2, $), where "$" unified with anything. If there were no existing bindings of parameters to constants, then both clauses in S would be deleted. From figure 13, to the contrary, we see that AT(OB1, P1) was not deleted; hence, it must have been the case that OB1 and OB2 represented distinct objects in the unparameterized problem for which the plan was originally created. If in a subsequent attempt to use this plan we set OB1 = OB2, then we are violating the constraint responsible for the occurrence of AT(OB1, P1)

in the final state. Accordingly, we replace the entry in cell (2,1) of Figure 13 by the new entry:

$$(OB1 \neq OB2) \supset AT(OB1,P1) \quad .$$

By this means we indicate that row 2, and cell (2,1) in particular, produces the literal AT(OB1, P1) only under the condition that OB1 and OB2 are not instantiated to the same constant.

The previous example illustrates how a literal can be allowed to survive the application of a delete function only under some condition of the bindings of its arguments. We introduced this notion in the context of maintaining the validity of a triangle table, but it is more broadly applicable within the general framework of STRIPS. Although it is an enlargement on our main theme of storing and using generalized plans, let us briefly consider how the notion of *conditional survival* of a literal can be exploited.

During the planning process, STRIPS frequently permits a delete function to delete true clauses from a state description. To overcome this tendency toward excessive deletions, we make use of the notion of conditional survival as defined by the following algorithm.

Let L(P1) be a literal in a parameterized state description, and suppose that the deletion of the clause containing this literal depends on binding parameter P1 to another parameter P2. Then:

- If P1 or P2 has no constant binding then replace L(P1) by $P1 \neq P2 \supset L(P1)$. (In "standard" STRIPS this clause would simply be deleted.)

- If P1 and P2 both represent the same constant in the original problem, then delete the clause containing L(P1). (This is what STRIPS does as a standard operation.) In the appropriate cell of the triangle table, place $P1 \neq P2 \supset L(P1)$. (This generalizes the triangle table beyond the planning states used by STRIPS.) If P1 and P2 represent distinct constants in the original problem, then replace L(P1) by $P1 \neq P2 \supset L(P1)$. (This is the case illustrated by our previous example.)

We should note that the inclusion in a table of such clauses as, say, $P1 \neq P2 \supset L(P1)$ leads to certain complications. Suppose, in a subsequent problem, that STRIPS uses such a clause in the proof of some precondition. Often, the proof will produce the unit clause

P1 = P2. In this case, we consider the proof completed by assuming P1 $\neq$ P2 (providing the assumption contradicts no existing bindings). However, we must record this assumption by placing P1 $\neq$ P2 in column 0 of the table being constructed; it is, after all, now a hypothesis of the theorem. Moreover, all subsequent proofs in the new plan must not violate this hypothesis. As a bookkeeping procedure, we can conjoin the assumption (viz., P1 $\neq$ P2) to each new precondition that STRIPS attempts to prove; this has the effect of preventing violations of our assumption.

b. Relaxing Preconditions in Nested Tables

Consider the situation shown in Figures 14(a) and (b), where we have shown a macro operator MOP whose i^{th} operator is itself the macro operator OP_i. As always, cell (i, i) of MOP contains the complete add-list of OP_i, while the marked entries of Row (i - 1) constitute the support of the proof of the preconditions of OP_i. During the planning process, suppose STRIPS selects from one of the rows of MOP certain clauses it would like to add to the current state of the world. Suppose further that some, but not all, of the clauses in cell (i,i) of Figure 14(a) are marked. We can therefore mark in Figure 14(b) those clauses in A_i that are needed, and exercise the operator extraction algorithm on table OP_i. As we saw earlier, this will at times result in the deletion of some of the clauses from PC_i. Suppose, then, that some of the clauses of PC_i are in fact deleted by the operator extraction algorithm. This raises the possibility of deleting some of the clauses in the support of PC_i since they now need to support only a weaker theorem. If the support of PC_i can be weakened—that is, if some of the clauses in row (i - 1) can be unmarked—than in general we may be able to delete more steps from MOP and/or obtain weaker, more easily established, preconditions for MOP.

In order for this scheme of precondition relaxation to be feasible, we need an economical solution to the following abstractly stated problem: Given that a set of clauses $C_1, ...,C_k$ implies a theorem $T_1 \cap ... \cap T_m$, which C_i's can be deleted from the premises if a selected subset of the T_i's are deleted from the theorem? Fortunately, it is possible to solve this problem by appropriately labeling literals during the refutation proof of the theorem. We will not elaborate here on the details of this bookkeeping procedure. In terms of the example of Figures 14(a) and (b) the important point is that the bookkeeping need be done only once, namely, when PC_i is shown to be a consequence of its support. Thereafter, it is a straightforward matter to compute, without recourse to theorem proving, the appropriate relaxation of the support of PC_i given a relaxation of PC_i itself.

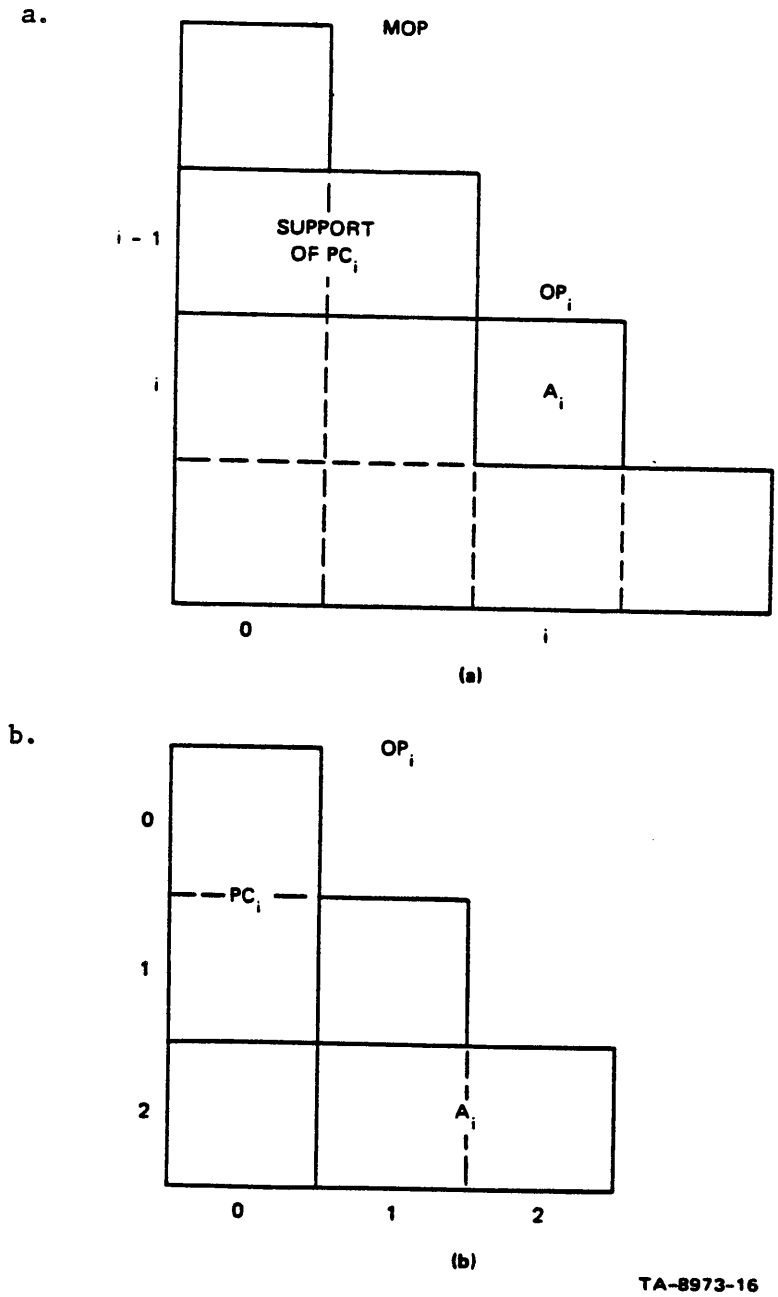


Figure 14: MOP: A NESTED MACROP*

**From [11], page 69.*

The ability to relax preconditions leads to an obvious refinement of the operator extraction algorithm described earlier. Previously, we unmarked clauses only when a component operator was deleted from a macro operator, in which case the entire support of the precondition of that operator was unmarked. Now we can also unmark a subset of the support of a component operator still retained in the macro operator. Finally, we remark that although Figure 14 shows only two levels of tables, the procedure for relaxing preconditions can be implemented recursively; hence; nested tables to arbitrary depth can be readily processed.

D. Monitoring the Execution of Plans

In this section we outline an algorithm for using triangle tables to monitor the real-world execution of generalized plans. An important feature of the algorithm is that it monitors only those effects of operators, and only those aspects of the world, relevant to the problem solution. Additionally, the algorithm embodies a modest replanning capacity in the form of an ability to reinstantiate parameters of operators.

The plan execution algorithm rests on the observation that a triangle table contains complete information about the internal structure of the plan it represents. More specifically, a triangle table specifies exactly what each operator accomplishes in terms of providing support for the preconditions of subsequent operators or the goal statement. Equivalently, a triangle table also specifies the conditions that must obtain in order for a component operator to be applicable.* The plan execution algorithm to be described uses this information in a straight-forward manner.

Important information about the internal structure of a plan is embodied in the *kernels* of a triangle table. The i^{th} kernel of a triangle table for an n-step plan is the largest rectangular subarray containing cells (n,0) and cell (i-1,i-1). In Figure 10, by way of an example, we have outlined the second kernel of MACROP. The importance of the i^{th} kernel stems from the fact that it contains the support of the preconditions for the tail of the plan—that is, the the operator sequence OP_i ,...,OP_n. This should be clear, since row j of the i^{th} kernel contains that portion of the support of PC_{j+1} that must already be true when OP_i is executed. To continue with the example of Figure 10, cells (2,0) and

*Strictly speaking, a triangle table specifies the support for the particular proof of a precondition found by STRIPS. In general, there are many possible supports for a given precondition, but we would not expect a plan execution algorithm to be cognizant of them.

(2,1) contain those axioms in PC_3 that are presumably true before OP_2 is executed. If any of these axioms are false, then even perfect execution of OP_2 will not result in a state in which OP_3 is applicable. Roughly speaking, then, a reasonable plan execution algorithm should find the highest indexed kernel with all true entries and execute the corresponding component operator.

Such an algorithm would reflect the heuristic that it is best to execute the "legal" operator least removed from the goal.

An important refinement of the rough execution algorithm outlined above can be obtained by noting that the ith kernel contains in general not only those clauses supporting proofs of preconditions but many additional clauses as well. These additional clauses are irrelevant to the problem at hand, and we would certainly want our execution algorithm to ignore them. The identification of relevant clauses is easily accomplished using the operator extraction algorithm previously described. The final row of the table representing a plan to be executed contains the support of the goal formula, and the supporting clauses are marked as before. The operator extraction algorithm then produces a new operator for achieving those clauses. (We dispense with the computation of precondition formula, add-list, and delete function.) Typically, but not necessarily, all the component operators will be retained. More importantly, only some of the table entries will be marked, and these are the only portions of the kernels that need be monitored.

The task of finding an efficient algorithm for finding the "highest true kernel"—that is, the highest indexed kernel with all marked clauses true—is of some interest in itself. Our algorithm for finding the highest true kernel involves a cell-by-cell scan of the triangle table. Each cell examined is evaluated as either True (i.e., all the marked clauses are true in the current model) or False. The interest of the algorithm stems from the order in which cells are examined. Let us call a kernel "potentially true" at some stage in the scan if all evaluated cells of the kernel are true. The scan algorithm can then be succinctly stated as:

> Among all unevaluated cells in the highest-indexed potentially true kernel, evaluate the left-most. Break "left-most ties" arbitrarily.

The reader can verify that, roughly speaking, this table-scanning rule results in a left-to-right, bottom-to-top scan of the table. However, the table is never scanned to the right of

any cell already evaluated as false. An equivalent statement of the algorithm is "Among all unevaluated cells, evaluate the cell common to the largest number of potentially true kernels. Break ties arbitrarily." We conjecture that this scanning algorithm is optimal in the sense that it evaluates, on the average, fewer cells than any other scan guaranteed always to find the highest true kernel. A proof of this conjecture has not been found.

The plan execution algorithm described above is embodied in a computer program named PLANEX [24]. When PLANEX is called to execute a table, it executes the component operator associated with the highest true kernel. Typically, but not necessarily, this will be OP_1. When OP_1 completes its action, PLANEX rescans the table to find the highest currently true kernel. Typically, but not necessarily, this will be Kernel 2, in which case OP_2 is executed, and so forth, until the goal kernel is reached. We emphasize, however, that after each operator execution PLANEX may either call an earlier operator (perhaps to rectify an error) or skip to a later operator (perhaps a stroke of luck rendered some operators unnecessary). Furthermore, many arguments of predicates in the table are parameters; PLANEX is free to instantiate these parameters in order to find a true instance of the predicate. Thus, PLANEX is really searching for the highest-indexed kernel an instance of which is satisfied by the current state of the world. This ability of PLANEX to instantiate—and reinstantiate—arguments provides a modest replanning capacity. If the turn of world events produces a situation in which a solution has the same form as a tail of the original plan, PLANEX will find it. If no tail of the plan is applicable, then no kernel will be true, and PLANEX returns control to STRIPS to replan.*

*From [11], pages 55-73.

CHAPTER NINE

Experiments With Shakey

In this final chapter we illustrate the capabilities described so far by giving Shakey some specific tasks. The material reprinted below (from [11]) is a description of planned experiments that were later carried out and recorded in a film and videotape available from SRI [25].

Experiments

In this section we shall describe some experiments now being planned that will illustrate several features of the robot system, which we call, informally, "Shakey." Specifically these will show how Shakey generates a plan to perform a task, and how it then uses part of this plan later as a component of a plan for performing another task. Saving plans for later use might be regarded as a form of learning. The experiments also show how the various levels in Shakey's hierarchical control structure function to enable Shakey to recover gracefully from several kinds of unexpected failures.

1. Shakey's World and Model

We must first describe the environment in which Shakey operates and Shakey's model of this environment. In Figure 15, we show a floor plan of some rooms and doorways in which our experiments with Shakey will be conducted. We can place several large boxes and wedge-shaped objects in these rooms; three boxes are depicted in room RCLK of Figure [15]. Initially Shakey is in room RUNI. The doorways all have mnemonic names indicating the rooms they connect; e.g., DMYSPDP connects RMYS and RPDP. Shakey's model of this environment is represented by a set of formulas or axioms in the first-order predicate calculus. The rooms, doorways, boxes, walls, and other entities occur as terms in formulas that describe important properties of the environment. The axiom model representing the environment for the planned experiments is listed in Table 6. The room names are mnemomics for properties of the physical environment:

RHAL = Hallway
RRIL = Rilla's office
RCLK = Room with the clock on the wall
RRAM = Room with ramp to hallway
RPDT = PDP-10 room
RUNI = Unimate room
RMYS = Mystery room, i.e., room with unknown contents.

The meanings of most of the predicate symbols are obvious. AT gives coordinate location information referenced to the coordinate system of Figure 15. DAT gives information about the probable error in this coordinate information. The RADIUS predicate is used to give rough size information. THETA and DTHETA give information about Shakey's heading and probable heading error, respectively. The UNBLOCKED predicate tells which doorways are unblocked (i.e., free of obstructing objects such as boxes). The predicate ROOMSTATUS is used to tell whether the contents of a room are known or unknown. The model listed in Table 6 indicates that the contents of all rooms are assumed to be known except for RMYS. By this we mean that Shakey *knows* that he will never encounter any new objects except perhaps in RMYS. This knowledge is used to guide certain picture-taking behavior, as we shall see later. The LANDMARKS predicate gives the locations of various landmarks such as corners and doorjambs that Shakey can take pictures of to update its position. The axioms at the end of the model in Table 6 (beginning with the predicate WHISKERS) give information about the status of various lower-level motor and sensing activities, e.g., the status of the catwhisker switches and camera control settings. (These were explained in Chapter Four.)

Altogether there are 170 axioms in the model initially, which makes this model quite large in comparison with those used by any previous automatic problem-solving systems.

2. Shakey's Action Repertoire

In order to perform the tasks described below, Shakey has available a repertoire of ILAs. (The operation of these ILAs is described in Chapter Five.) The problem-solving system, STRIPS, must be aware of the properties of the available ILAs. Therefore each ILA is represented for STRIPS by an operator with specified preconditions and effects. These operators and their descriptions are given in Table 7 using the add and delete lists employed by STRIPS.

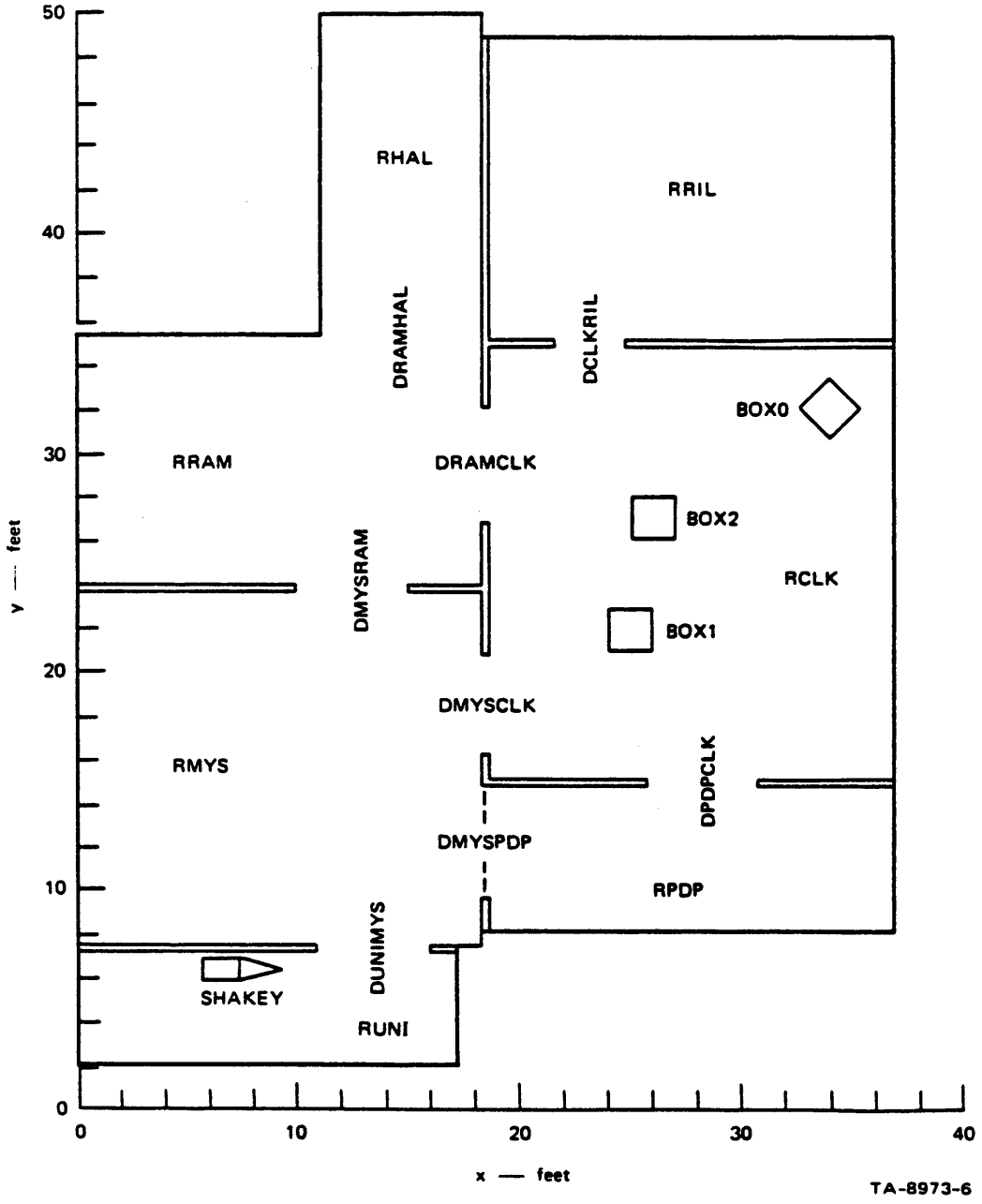


Figure 15: MAP OF SHAKEY'S EXPERIMENTAL ENVIRONMENT*

*From [11], page 6.

```
AT(ROBOT,7,5)
DAT(ROBOT,0.1,0.1)
INROOM(ROBOT,RUNI)
AT(BOX0,34,32)
INROOM(BOX0,RCLK)
AT(BOX1,25,22)
INROOM(BOX1,RCLK)
AT(BOX2,26,27)
INROOM(BOX2,RCLK)
SHAPE(BOX0,BOX)
SHAPE(BOX1,BOX)
SHAPE(BOX2,BOX)
RADIUS(BOX0,1.7)
RADIUS(BOX1,1.5)
RADIUS(BOX2,1.5)
DAT(BOX0,0.1)
DAT(BOX1,0.1)
DAT(BOX2,0.1)
THETA(ROBOT,-90)
DTHETA(ROBOT,1)
PUSHABLE(BOX1)
PUSHABLE(BOX2)
UNBLOCKED(DRAMHAL,RHAL)
UNBLOCKED(DRAMHAL,RRAM)
UNBLOCKED(DCLKRIL,RRIL)
UNBLOCKED(DCLKRIL,RCLK)
UNBLOCKED(DRAMCLK,RCLK)
UNBLOCKED(DRAMCLK,RRAM)
UNBLOCKED(DMYSRAM,RMYS)
UNBLOCKED(DMYSRAM,RRAM)
UNBLOCKED(DMYSCLK,RCLK)
UNBLOCKED(DMYSCLK,RMYS)
UNBLOCKED(OPOPCLK,RCLK)
UNBLOCKED(DPDPCLK,RPDP)
UNBLOCKED(DMYSPDP,RPDP)
UNBLOCKED(DMYSPDP,RMYS)
UNBLOCKED(DUNIMYS,RMYS)
UNBLOCKED(DUNIMYS,RUNI)
BOUNDSROOM(FSRAM RRAM SOUTH)
BOUNDSROOM(FERAM RRAM EAST)
BOUNDSROOM(FWRAM RRAM WEST)
BOUNDSROOM(FNCLK RCLK NORTH)
BOUNDSROOM(FSCLK RCLK SOUTH)
BOUNDSROOM(FECLK RCLK EAST)
BOUNDSROOM(FWCLK RCLK WEST)
BOUNDSROOM(FNMYS RMYS NORTH)
BOUNDSROOM(FSMYS RMYS SOUTH)
BOUNDSROOM(FEMYS RMYS EAST)
BOUNDSROOM(FWMYS RMYS WEST)
BOUNDSROOM(FNPDP RPDP NORTH)
BOUNDSROOM(FSPDP RPDP SOUTH)
BOUNDSROOM(FEPDP RPDP EAST)
BOUNDSROOM(FWPDP RPDP WEST)
BOUNDSROOM(FNUNI RUNI NORTH)
BOUNDSROOM(FSUNI RUNI SOUTH)
BOUNDSROOM(FEUNI RUNI EAST)
BOUNDSROOM(FWUNI RUNI WEST)
FACELOC(FNHAL 50.0)
FACELOC(FSHAL 35.5)
FACELOC(FEHAL 18.200000)
FACELOC(FWHAL 11.200000)
FACELOC(FNRIL 49.0)
```

Table 6: AXIOM MODEL

```
FACELOC(FSRIL 35.400000)
FACELOC(FERIL 36.800000)
FACELOC(FWRIL 18.799998)
FACELOC(FNRAM 35.5)
FACELOC(FSRAM 24.0)
FACELOC(FERAM 18.200000)
FACELOC(FWRAM 0.0)
FACELOC(FNCLK 35.0)
FACELOC(FSCLK 15.200000)
FACELOC(FECLK 36.800000)
FACELOC(FWCLK 18.599997)
FACELOC(FNMYS 23.599997)
FACELOC(FSMYS 7.6000000)
FACELOC(FEMYS 18.200000)
FACELOC(FWMYS 0.0)
FACELOC(FNPDP 14.799998)
FACELOC(FSPDP 8.2000000)
FACELOC(FEPDP 36.800000)
FACELOC(FWPDP 18.600000)
FACELOC(FNUNI 7.1999999)
FACELOC(FSUNI 2.1999998)
FACELOC(FEUNI 17.200000)
FACELOC(FWUNI 0.0)
JOINSROOMS(DRAMHAL RRAM RHAL)
JOINSROOMS(DRAMCLK RRAM RCLK)
JOINSROOMS(DCLKRIL RCLK RRIL)
JOINSROOMS(DRAMHAL RHAL RRAM)
JOINSROOMS(DRAMCLK RCLK RRAM)
JOINSROOMS(DCLKRIL RRIL RCLK)
TYPE(BOX1 OBJECT)
TYPE(BOX2 OBJECT)
TYPE(BOX0 OBJECT)
TYPE(RHAL ROOM)
TYPE(RRIL ROOM)
TYPE(RRAM ROOM)
TYPE(RCLK ROOM)
TYPE(RMYS ROOM)
TYPE(RPDP ROOM)
TYPE(RUNI ROOM)
TYPE(DRAMHAL DOOR)
TYPE(DRAMCLK DOOR)
TYPE(DCLKRIL DOOR)
TYPE(DMYSRAM DOOR)
TYPE(DMYSCLK DOOR)
TYPE(DMYSPDP DOOR)
TYPE(DPDPCLK DOOR)
TYPE(DUNIMYS DOOR)
BOUNDSROOM(FNHALL RHAL NORTH)
BOUNDSROOM(FSHAL RHAL SOUTH)
BOUNDSROOM(FEHAL RHAL EAST)
BOUNDSROOM(FWHAL RHAL WEST)
BOUNDSROOM(FNRIL RRIL NORTH)
BOUNDSROOM(FSRIL RRIL SOUTH)
BOUNDSROOM(FERIL RRIL EAST)
BOUNDSROOM(FWRIL RRIL WEST)
BOUNDSROOM(FNRAM RRAM NORTH)
JOINSROOMS(DMYSRAM RNYS RRAM)
JOINSROOMS(DMYSCLK RMYS RCLK)
JOINSROOMS(DMYSPDP RMYS RPDP)
JOINSROOMS(DPDPCLK RPDP RCLK)
JOINSROOMS(DUNIMYS RUNI RMYS)
JOINSFACES(DRAMHAL FNRAM FSHAL)
JOINSFACES(DRAMCLK FERAM FWCLK)
```

TABLE: 6, continued

```
JOINSFACES(DCLKRIL FNCLK FSRIL)
JOINSFACES(DMYSRAM FNMYS FSRAM)
JOINSFACES(DMYSCLK FEMYS FWCLK)
JOINSFACES(DMYSPDP FEMYS FWPDP)
JOINSFACES(DPDPCLK FNPDP FSCLK)
JOINSFACES(DUNIMYS FNUNI FSMYS)
DOORLOCS(DRAMHAL 11.200000 18.200000)
DOORLOCS(DRAMCLK 26.799998 32.0)
DOORLOCS(DCLKRIL 21.700000 24.799998)
DOORLOCS(DMYSRAM 10.0 15.200000)
DOORLOCS(DMYSCLK 16.200000 20.799998)
DOORLOCS(DMYSRDP 9.7000000 14.799998)
DOORLOCS(DPDPCLK 25.799998 30.799998)
DOORLOCS(DUNIMYS 10.799998 16.0)
ROOMSTATUS(RHAL KNOWN)
ROOMSTATUS(RRIL KNOWN)
ROOMSTATUS(RRAM KNOWN)
ROOMSTATUS(RCLK KNOWN)
ROOMSTATUS(RMYS UNKNOWN)
ROOMSTATUS(RPDP KNOWN)
ROOMSTATUS(RUNI KNOWN)
LANDMARKS(RHAL (COORDS (4. 11.200000 35.5 0.)))
LANDMARKS(RRIL
          (COORDS (4. 21.700000 35.400000 -1.)
                  (3. 24.799998 35.400000 -1.)
                  (2. 18.799998 49.0 4.)
                  (2. 36.800000 49.0 3.)
                  (2. 36.800000 35.400000 2.)
                  (2. 18.799998 35.400000 1.)))
LANDMARKS(RRAN
          (COORDS (4. 18.200000 26.799998 0.)
                  (3. 18.200000 32.0 0.)
                  (1. 11.200000 35.5 2.)
                  (4. 10.0 24.0 -1.)
                  (3. 15.200000 24.0 -1.)
                  (2. 0.0 35.5 4.)
                  (2. 18.200000 24.0 2.)
                  (2. 0.0 24.0 1.)))
JOINSROOMS(DMYSRAM RRAM RMYS)
JOINSROOMS(DMYSCLK RCLK RMYS)
JOINSROOMS(DMYSPDP RPDP RMYS)
JOINSROOMS(DPDPCLK RPDP RCLK)
JOINSROOMS(DUNIMYS RUNI RMYS)
LANDMARKS(RCLK
          (COORDS (4. 24.799998 35.0 -1.)
                  (3. 21.700000 35.0 -1.)
                  (4. 25.799998 15.200000 -1.)
                  (3. 30.799998 15.200000 -1.)
                  (4. 18.599997 20.799998 0.)
                  (3. 18.599997 16.200000 0.)
                  (4. 18.599997 32.0 0.)
                  (3. 18.599997 26.799998 0.)
                  (2. 18.599997 35.0 4.)
                  (2. 36.800000 35.0 3.)
                  (2. 36.800000 15.200000 2.)
                  (2. 18.599997 15.200000 1.)))
LANDMARKS(RMYS
          (COORDS (4. 18.200000 9.7000000 4.)
                  (1. 18.200000 14.799998 1.)
                  (4. 18.200000 16.200000 0.)
                  (3. 18.200000 20.799998 0.)
                  (4. 15.200000 23.599997 -1.)
                  (3. 10.0 23.599997 -1.)
```

TABLE 6, continued

```
                (4. 10.799998 7.6000000 -1.)
                (3. 16.000000 7.6000000 -1.)
                (2. 0.0 23.599997 4.)
                (2. 18.200000 23.599997 3.)
                (2. 18.200000 7.6000000 2.)
                (2. 0.0 7.6000000 1.)))
LANDMARKS(RPDP
          (COORDS (4. 30.799998 14.799998 -1.)
                (3. 25.799998 14.799998 -1.)
                (4. 18.200000 14.799998 -1.)
                (3. 18.600000 9.7000000 0.)
                (2. 36.800000 14.799998 3.)
                (2. 36.800000 8.2000000 2.)))
LANDMARKS(RUNI
          (COORDS (4. 16.000000 7.1999999 -1.)
                (3. 10.799998 7.1999999 -1.)
                (2. 16.0 7.1999999 3.0)
                (2. 17.200000 2.1999998 2.)
                (2. 0.0 2.1999998 1.)))
WHISKERS(ROBOT,0)
IRIS(ROBOT,1)
OVERIDE(ROBOT,0)
RANGE(ROBOT,30)
TVMODE(ROBOT,0)
FOCUS(ROBOT,30)
PAN(ROBOT,0)
TILT(ROBOT,0)
DPAN(ROBOT,3.12)
DTILT(ROBOT,0.7)
DIRIS(ROBOT,0)
DFOCUS(ROBOT,0)
PICTURESTAKEN(ROBOT,0)
JUSTBUMPED(ROBOT,NIL)
```

TABLE 6, concluded

We shall now describe the planned experiments that will use the model of Table 6 and the operators shown in Table 7. The description will be in terms of the expected results of these experiments.

a. Task 1

Starting with the configuration of Figure 15 (represented by the model in Table 6), Shakey will perform two tasks. Each of these tasks is stated in English and entered into the system via teletype. The first task is stated as "USE BOX 2 TO BLOCK DOOR DPDPCLK FROM ROOM RCLK." This statement is converted by the English language system ENGROB [26] to a goal expressed by a well-formed formula (wff) of the first-order predicate calculus: BLOCKED(DPDPCLK,RCLK,BOX2). The STRIPS problem-solving system is then called to compose a sequence of operators whose execution will create a world model in which this goal wff is true. In terms of the operators in Table 7, we can show that the following sequence would solve this problem:

GOTO2(DUNIMYS),GOTHRUDR(DUNIMYS,RUNI,RMYS),
GOTO2(DMYSCLK),
GOTHRUDR(DMYSCLK,RMYS,RCLK),
BLOCK(DPDPCLK,RCLK,BOX2) .

Rather than generating this specific solution, STRIPS generates a *generalized* plan that involves going from an arbitrary initial room through an intermediate room, and into a third room and then blocking a doorway in the third room. The rooms, doorways, and blocking object in this generalized plan are represented by *parameters*. The generalized plan is thus a subroutine whose arguments are the parameters. These arguments are bound to specific constants only when the plan is executed. The value of the generalized subroutine is that it can be stored away (or "learned") and then used again in other situations perhaps as part of a plan for a more complex task.

BLOCK(DX,RX,BX)

Preconditions:

INROOM(ROBOT,RX) ∧ INROOM(BX,RX)
∧ PUSHABLE(BX) ∧ UNBLOCKED(DX,RX)
∧ (∃RY)JOINSROOMS(DX,RX,RY)

Delete List:

AT(ROBOT,$1,$2)
AT(BX,$1,$2)
UNBLOCKED(DX,RX)
NEXTTO(ROBOT,$1)
NEXTTO(BX,$1)
NEXTTO($1,BX)

Add List:

*BLOCKED(DX,RX,BX)
NEXTTO(ROBOT,BX)

Blocks door DX with an object BX by pushing BX to a place in room RX directly in front of door DX.

UNBLOCK(DX,RX,BX)

Preconditions:

BLOCKED(DX,RX,BX) ∧ INROOM(ROBOT,RX) ∧ PUSHABLE(BX)

Delete List:

AT(ROBOT,$1,$2)
BLOCKED(DX,RX,BX)
AT(BX,$1,$2)
NEXTTO(ROBOT,$1)
NEXTTO(BX,$1)
NEXTTO($1,BX)

Add List:

*UNBLOCKED(DX,RX)
NEXTTO(ROBOT,BX)

Unblocks door DX by pushing object BX away from its place in room RX directly in front of door DX.

GOTHRUDR(DX,RX,RY)

Preconditions:

NEXTTO(ROBOT,DX) ∧ INROOM(ROBOT,RX)
∧ JOINSROOMS(DX,RX,RY) ∧ UNBLOCKED(DX,RX)
∧ UNBLOCKED(DX,RY)

Delete List:

AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)
INROOM(ROBOT,$1)

Table 7: STRIPS OPERATORS

Add List:

*INROOM(ROBOT,RY)
NEXTTO(ROBOT,DX)

Takes Shakey through door DX from room RX into room RY.

GOTO2(X)

Preconditions:

(∃RX)[INROOM(ROBOT,RX) ∧ INROOM(X,RX)]
∨ (∃RX,RY)[INROOM(ROBOT,RX)
∧ JOINSROOMS(X,RX,RY) ∧ UNBLOCKED(X,RX)]

Delete List:

AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)

Add List:

*NEXTTO(ROBOT,X)

Takes Shakey from any point in a room to a location next to any object or doorway, X, in the same room. (Shakey will navigate around obstacles that might be in the way of a direct path.)

PUSH(OB,X,Y)

Preconditions:

(∃RX)[INROOM(ROBOT,RX) ∧
INROOM(OB,RX) ∧ LOCINROOM(X,Y,RX)]
∧ PUSHABLE(OB)

Delete List:

AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)
AT(OB,$1,$2)
NEXTTO(OB,$1)
NEXTTO($1,OB)

Add List:

*AT(OB,X,Y)
NEXTTO(ROBOT,OB)

Pushes object OB from one point in a room to a coordinate location (X,Y) in the same room. (Shakey must initially be in the same room as OB and (X,Y), but will push OB around obstacles that might be in the way of a direct path.)

NAVTO(X,Y)

Preconditions:

(∃RX)[INROOM(ROBOT,RX)
∧ LOCINROOM(X,Y,RX)]

TABLE 7, continued

Delete List:

AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)

Add List:

*AT(ROBOT,X,Y)

Takes Shakey from any point in a room to the coordinate location (X,Y) in the same room. (Shakey will navigate around obstacles that might be in the way of a direct path.)

POINT(DIRECTION)

Preconditions:

none

Delete List:

THETA(ROBOT,$1)

Add List:

*THETA(ROBOT,DIRECTION)

Turns Shakey so that its heading is DIRECTION.

PUSH3(OB,X)

Preconditions:

PUSHABLE(OB) ∧ ∃(RX)[INROOM(ROBOT,RX) ∧ INROOM(OB,RX)
∧ [INROOM(X,RX) ∨ ∃(RY)JOINSROOMS(X, RX,RY)]]

Delete List:

AT(ROBOT,$1,$2)
NEXTTO(ROBOT,$1)
AT(OB,$1,$2)
NEXTTO(OB,$1)
NEXTTO($1,OB)

Add List:

*NEXTTO(OB,X)
NEXTTO(ROBOT,OB)

Pushes object OB from one point in a room to a location next to any object or doorway X in the same room. (Shakey will push OB around obstacles that might be in the way of a direct path.)

*Note: An asterisk(*) in front of an add-list clause indicates that this clause is one of the "primary effects" of the operator.

TABLE 7, concluded**

**From [11], pages 13-15.

The task in question elicits the following generalized plan from STRIPS:

GOTO2(PAR6),GOTHRUDR(PAR6,PAR7,PAR5,)
GOTO(PAR4),GOTHRUDR(PAR4,PAR5,PAR2),
BLOCK(PAR1,PAR2,PAR3) .

This plan is stored away as the macro operator:

MACROP1(PAR3,PAR1,PAR2,PAR4,PAR5,PAR7,PAR6) .

STRIPS creates a triangle table representation of MACROP1. This table compactly stores information vital to monitoring the execution of MACROP1 and information needed to use MACROP1 (or parts of it) as a component of a future plan. We show this triangle table representation of MACROP1 in Table 8* and refer the reader to Chapter Eight for a discussion of triangle tables and their uses.

After the creation of the triangle table representation of MACROP1, STRIPS prepares a version of it that will solve the given task, namely, to "Use BOX2 to block door DPDCLK from room RCLK." This version is obtained from MACROP1 by replacing those parameters standing for constants in the goal wff by those constants. That is, in this case, we replace PAR1 by DPDPCLK, PAR2 by RCLK, and PAR3 by BOX2 throughout the MACROP1 triangle table. This instantiated table is then given to PLANEX for execution.

PLANEX is a program that supervises the execution of those ILAs corresponding to the operators in the plan. For a discussion of the operation of PLANEX, see the last part of Chapter Eight. PLANEX takes as input a partially instantiated MACROP in triangle table form. (This MACROP may have some parameters remaining after those occurring in the goal wff have been instantiated.) The PLANEX algorithm looks for a specific, fully instantiated subsequence of the operators in the MACROP that can be executed in the present situation to achieve the goal. The ILA corresponding to the first operator is then executed. In the case of the task we are considering the first ILA to be executed is GOTO2(DUNIMYS), which causes the robot to go to the door named DUNIMYS.

---

*Note: For all triangle tables, an asterisk (*) before a clause indicates that this clause was used to prove the preconditions of the operator named at the right of the row in which the clause appears.

The PLANEX algorithm then determines that the next ILA to be executed should be GOTHRUDR(DUNIMYS,RUNI,RMYS). Execution of this ILA begins by calling the vision routine CLEARPATH, which takes a TV picture through the doorway to determine whether the path in RMYS is clear (since the contents of RMYS are unknown). The path is in fact clear, so Shakey proceeds through the doorway.

Next PLANEX calls for the execution of GOTO2(DMYSCLK). Since the contents of RMYS are unknown to Shakey, GOTO calls CLEARPATH again. To illustrate how Shakey can deal with unforeseen difficulties, we now place a box directly in Shakey's path in front of the door DMYSCLK. As Figure 15 and Table 6 show, Shakey does not know of the existence of this box. CLEARPATH determines that the path is blocked and notes the approximate location of the blocking object. Since Shakey expects that it might encounter unknown objects in room RMYS, GOTO next calls a vision routine called OBLOC. This routine calculates the size and exact location of the object, gives it a name, BOX3, and adds this information to the model. (it also assumes, perhaps optimistically, that the new box is pushable.) OBLOC also notes that BOX3 is blocking door DMYSCLK, so it adds the wff BLOCKED(DMYSCLK,RMYS,BOX3) to the model. Since the conditions for continuing the execution of GOTO(DMYSCLK) are no longer satisfied, control returns to PLANEX. Our interest in this experiment is to show how Shakey can gracefully recover from such an unexpected failure of its plan.

PLANEX, as usual, attempts to find a fully instantiated version of the parameterized MACROP1 that can be executed in the present situation to achieve the goal. In this case, PLANEX finds another instantiation of MACROP1 that works. The operators in this instantiation are:

GOTO2(DMYSPDP),GOTHRUDR(DMYSPDP,RMYS,RPDP),
GOTO2(DPDPCLK),
GOTHRUDR(DPDPCLK,RPDP,RCLK)
BLOCK(DPDPCLK,RCLK,BOX2).

Here we see one of the advantages of constructing parameterized plans. To perform the original task, we first constructed a parameterized plan having an instance that solves the problem. Later in the task execution we find that after an unexpected difficulty, another instance of the same parameterized plan can be used to achieve the goal. We expect that this method of error recovery will be quite valuable in robot problems. (If PLANEX could

Table 8: TRIANGLE TABLE FOR MACROP1(PAR3,PAR1,PAR2,PAR4,PAR5,PAR7,PAR6)*

TRIANGLE TABLE FOR MACROP1(PAR3,PAR1,PAR2,PAR4,PAR5,PAR7,PAR6)

*UNBLOCKED(PAR6,PAR7) *JOINSROOMS(PAR6,PAR7,PAR5) *INROOM(ROBOT,PAR7)	GOTO2(PAR6)			
*UNCLOCKED(PAR6,PAR7) *INROOM(ROBOT,PAR7) *JOINSROOMS(PAR6,PAR7,PAR5) *UNBLOCKED(PAR6,PAR5)	*NEXTTO(ROBOT,PAR6)	GOTHRUDR(PAR6,PAR7,PAR5)		
*UNBLOCKED(PAR4,PAR5) *JOINSROOMS(PAR4,PAR5,PAR2)		*INROOM(ROBOT,PAR5) NEXTTO(ROBOT,PAR6)	GOTO2(PAR4)	
*UNBLOCKED(PAR4,PAR5) *JOINSROOMS(PAR4,PAR5,PAR2) *UNBLOCKED(PAR4,PAR2)		*INROOM(ROBOT,PAR5)	*NEXTTO(ROBOT,PAR4)	GOTHRUDR(PAR
*~EQ(ROBOT,PAR3) *UNBLOCKED(PAR1,PAR2) *~EQ(ROBOT,PAR3)⇒INROOM(PAR3,PAR2) *PUSHABLE(PAR3) *JOINSROOMS(PAR1,PAR2,PAR8)				*INROOM(ROBO NEXTTO(ROBOT
				INROOM(ROBOT

*From [11], page 17.

find no applicable instance of MACROP1 that would achieve the goal, then STRIPS would be asked to produce another plan and MACROP.)

After finding this new instance of MACROP1, PLANEX calls for the execution of the first operator GOTO2(DMYSPDP). Shakey thus moves to door DMYSPDP. PLANEX next calls for going through the door, and the process continues until finally Shakey enters room RCLK. Then PLANEX calls for the execution of BLOCK(DPDPCLK,RCLK,BOX2). Running this ILA calls for going to BOX2 and pushing it around BOX1 and then to door DPDPCLK (a "two-leg" push). The local planning needed to accomplish this push operation is done entirely within the PUSH ILA called by BLOCK. With this operation complete, Shakey has accomplished the first task, in spite of the unforeseen difficulty. We also note that MACROP1 has been filed away and can be used as an operator in future problem solving.

b. Task 2

The state of things in Shakey's world is now as shown in Figure 16. We now test Shakey's ability to learn by giving it a task that can be solved by using *part* of MACROP1. The statement of the task given to the system, in English, is "UNBLOCK DOOR DYMSCLK FROM ROOM RMYS." That is, we want Shakey to move away the object (BOX3) that it discovered to be blocking DMYSCLK.

Again, the English statement is converted into a predicate calculus wff:

UNBLOCKED(DMYSCLK,RMYS).

STRIPS now attempts to find a sequence of operators that will make the wff true, but now it has MACROP1 available in its operator repertoire (in addition to the operators corresponding to ILAs). STRIPS first decides that it should try to apply the operator UNBLOCK(DMYSCLK,RMYS,BOX3). To do so, Shakey must be in room RMYS, so STRIPS looks for operators that will achieve INROOM(ROBOT,RMYS).

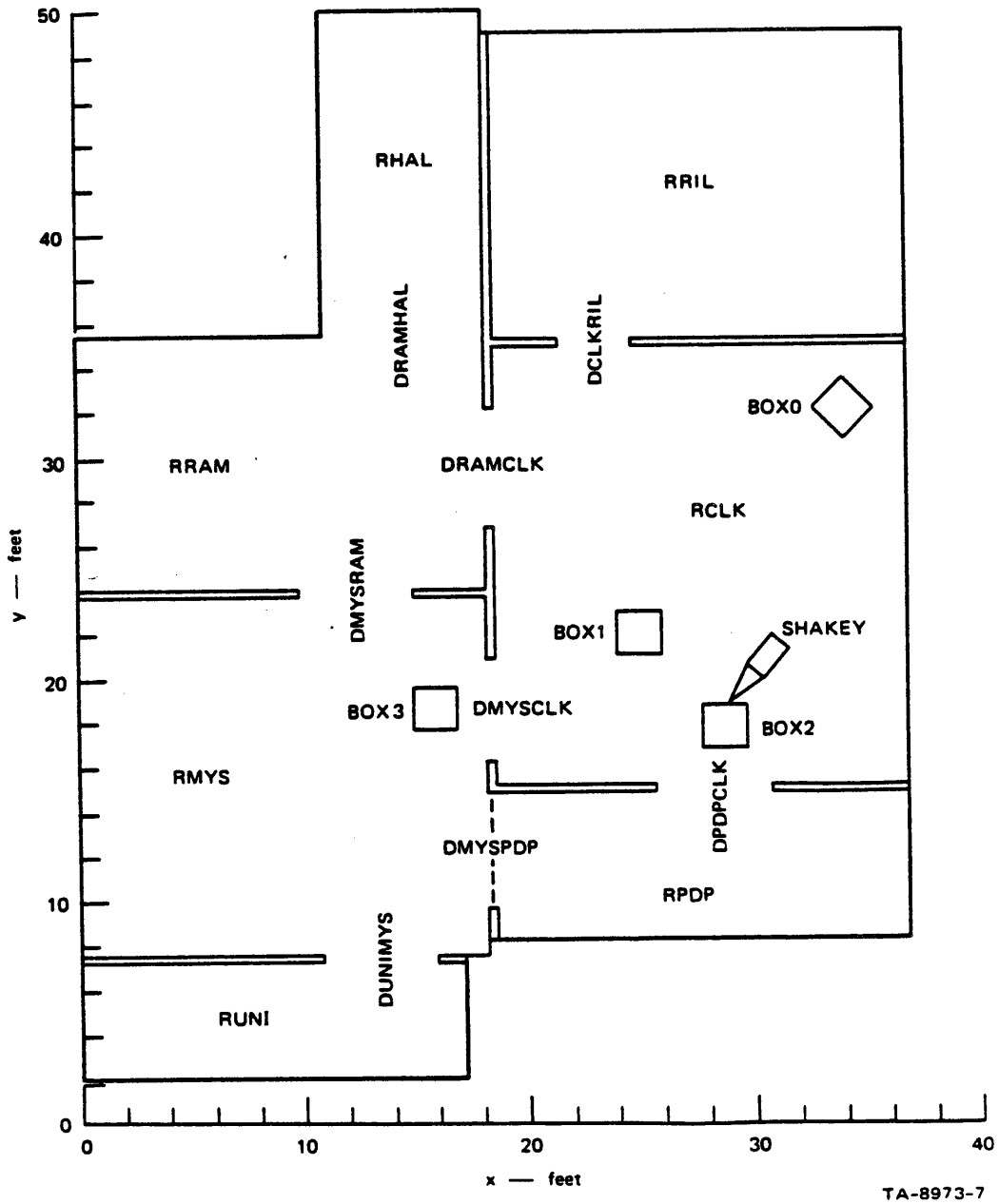


Figure 16: MAP OF SHAKEY'S WORLD AFTER COMPLETION OF THE FIRST TASK*

*From [11], page 21,

situation and achieves the condition INROOM(ROBOT,RMYS), STRIPS is quickly able to settle on this instance and produce a plan for Task 2. Let us denote by MACROP1′ the subsequence of MACROP1 selected by STRIPS. MACROP1′ still contains free parameters that are left to be bound at execution time. Its definition in terms of the operators comprising it is:

MACROP1′ (PAR2,PAR4,PAR5,PAR7,PAR6)

GOTO2(PAR6)
GOTHRUDR(PAR6,PAR7,PAR5)
GOTO2(PAR4)
GOTHRUDR(PAR5,PAR2) .

The complete generalized plan for the second ask is:

MACROP1′ (PAR2,PAR4,PAR5,PAR7,PAR6)
UNBLOCK(PAR1,PAR2,PAR3) .

This generalized plan is given the name MACROP2 and is saved for possible later use. The triangle table representation of MACROP2 is shown in Table 8.

After creating the general version of MACROP2, STRIPS prepares a version of it for PLANEX by instantiating it with those constants appearing in the task description. Namely, DMYSCLK is substituted for PAR1 and RMYS for PAR2. It then gives this partially instantiated version to PLANEX to be executed. PLANEX finds that the following instantiation of the plan will achieve the goal:

MACROP1′ (RMYS,DMYSRAM,RRAM,RCLK,DRAMCLK)
UNBLOCK(DMYSCLK,RMYS,BOX3) .

Next, PLANEX calls for execution of MACROP1′. This execution is accomplished by PLANEX itself. The ability to handle "nested" triangle tables is one of the features of our system. PLANEX discovers that the first ILA to be executed in MACROP1′ is GOTO(DRAMCLK). In a similar manner, PLANEX ultimately executes the entire string of ILAs in MACROP1′ and then the UNBLOCK ILA to accomplish the second task.

Table 9: TRIANGLE TABLE FOR MACROP2(PAR3,PAR1,PAR6,PAR7,PAR5,PAR4,PAR2)*

TRIANGLE TABLE FOR MACROP2(PAR3,PAR1,PAR6,PAR7,PAR5,PAR4,PAR

*INROOM(ROBOT,PAR7) *UNBLOCKED(PAR6,PAR7) *JOINSROOMS(PAR6,PAR7,PAR5) *UNBLOCKED(PAR6,PAR5) *JOINSROOMS(PAR4,PAR5,PAR2) *UNBLOCKED(PAR4,PAR5) *UNBLOCKED(PAR4,PAR2)	MACROP1'(PAR2,PAR4,PAR5,PAR7,PAR6)	
*BLOCKED(PAR1,PAR2,PAR3)	*INROOM(ROBOT,PAR2) NEXTTO(ROBOT,PAR4)	UNBLOCK(PAR1
	INROOM(ROBOT,PAR2)	NEXTTO(ROBOT UNBLOCKED(PA

*From [11], page 29.

When these experiments are actually conducted, it is probable that the system may decide to exercise another one of our error-recovery capabilities. Recall that the model contains information about the probable error in Shakey's location stored in the predicate DAT. Model-maintenance programs automatically increase the estimate of error after every robot motion. During execution of ILAs such as GOTO2, this probable error is checked to see whether it is still less than some specific tolerable error. Whenever the error estimate exceeds the tolerance, a visual program called LANDMARK is called. LANDMARK takes a picture of some nearby feature (such as a joorjamb), calculates from this picture the robot's actual location, and enters this updated location into the model. It also resets the DAT predicate to the error estimate appropriate after having just taken a picture.

Several features of the system are illustrated in these experiments. Most important of these are the ability to learn generalized plans and the ability to recover from various types of failures. The system of ILAs is designed to be robust in the sense that each ILA does what it can locally to correct any errors. When the appropriate recovery procedures are beyond a specific ILA's knowledge and abilities, there are several higher levels where recovery can occur, namely, at higher level ILAs, in PLANEX, or in STRIPS.*

ACKNOWLEDGMENTS

Many people worked on the Shakey project. Charles A. Rosen, the founder of the SRI Artificial Intelligence Center, first conceived of "an intelligent automaton project." In an attempt to mention at least some researchers, we have explicitly listed in the references to this note all of the authors of the Shakey technical reports (instead of using the usual "et al." convention.) We also gratefully acknowledge the Defense Advanced Research Projects Agency who supported the research described here. A special note of appreciation is due Dr. Ruth Davis who, as a senior official in the Defense Department of Research and Engineering, had the vision to initiate this and other projects in robotics.

Appendix A

Mechnical Development of the Automaton Vehicle

Appendix A

Mechnical Development of the Automaton Vehicle

By Vladimir Lieskovsky

The following note from [3] by Vladimir Lieskovsky described the robot vehicle in some detail:

MECHANICAL DEVELOPMENT OF THE AUTOMATON VEHICLE

A. General Arrangement of the Vehicle

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

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

between the wheels accommodates the main drive motors, and for a low center of gravity, the batteries.

A 4-in vertical distance above the platform was reserved for proposed manipulator arms. A standard 19-in electronic rack, supported at three points, was located above this reserved space. A video camera and range finder combination was mounted atop the rack.

B. Details of the Physical Arrangement

1. Power Supply and Drive

One of the first decisions to be made was the selection of the form of energy to be used for drive purposes. Among those considered were hydraulic, pneumatic, and eventually, electric drives. Since electrical power had to be made available for the electronics, electric drive was ultimately selected. The choice between secondary batteries and fuel cells was dictated mainly by price and delivery figures in favor or the batteries. Two 12-volt batteries in series were used to establish the operational, nominal voltage at 24 Vdc. The choice between drive motors was reduced to either a straight dc motor, an inverter and ac motor combination, or stepping motors. Complexity and control considerations of the digital commands ruled out the inverter/ac combination. Direct current motors, although electrically noisy, were attractive due to their high power density and good torque characteristics. Manufacturer's quotes were uniformly forbidding: six months for delivery and a price in excess of several thousand dollars for each motor. The units would have had standard clutches, brakes, and position readout capability for feedback information. Stepping motors, although they suffer from low power density, are excellently suited for digital control, and they were immediately available and were low in price (not more than about $200.00 each). Therefore, the decision was made to use stepping motors exclusively for prime movers. Not all of the motors selected were rated at 24 Vdc, but they were easily converted by using dropping resistors.

In order not to lose count of the steps in the drive train between the motor and the drive wheel, the speed reduction between the motor and the wheels had to be one without slippage, that is, positive. The reduction was necessary to increase available torque from the motors and to reduce the amount of translation per incremental step of the motor to

1/32nd of an inch measured at the periphery of the wheel. For every control pulse, the stepping motor executes a rapid change in its angular position. Depending on the inertia of the driven load and the damping of the drive trains, oscillations may develop. These oscillations were reduced by limiting the incremental stepsize, i.e., the generated amplitude. A cogged belt, or *timing belt*, arrangement was selected for the drive train. This was to give the necessary positive drive, while also introducing damping. As it turned out, the belt proved to be a secondary source of oscillations, since bending vibrations were generated in the belt when the stepping motor was operated. Increasing the belt tension reduced the oscillations to an acceptable level.

2. Closing the Minor Loop Through the Motor

The stepping motor operates in an open loop mode. Completion of any step depends on the inertial load coupled to the motor, and not unlike a synchronous motor, the stepping motor also can "fall out of phase," so to say, when it is overloaded. This condition is largely a function of the stepping rate. Therefore, closing the loop in the operation of the main drive motors seemed to be warranted. Fortunately, similar considerations led Fredrikson [27] to synthesize, build, and describe a closed-loop stepping motor scheme. By using his results, we were able to adhere to the ground rule of no novel detail development. We closed the minor loop through the motor in the following way: a disk, containing fifty appropriate holes on a circle, was mounted on the motor shaft. Four light source and photocell pairs placed along the circle, and shifted by one-fourth of the hole pattern pitch, were mounted on the motor housing. This arrangement provided for 200 positions for every revolution, which is also the step-pattern of the motor. We used the simple schematic, described in [27] to complete the feedback loop. In operation, no step command can be given until after the information from the position feed-back disk indicates that the previous step has been completed. Simply, the motor cannot miss a step.

3. Wheels

The rubber wheels presented another problem: due to their finite elasticity, transient motions generated either by the vehicle itself, or by its environment, resulted in disturbing oscillations of the whole vehicle in pitch and roll modes with a time constant of about 2 seconds. This amount of settling time was judged to be unacceptable because no picture taking with the TV camera could be initiated during that time. Since friction on the driving wheels had to be maintained, but elasticity minimized, a properly-stiffened rubber

driving rim on a metal wheel proved to be an acceptable solution. Since the castor wheels, however, could remain relatively compliant, but required reduced friction on the floor, they were capped with a metallic rim and gave good results.

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

4. TV Camera and Range Finder Mount

Although it is possible to scan with a TV camera which is rigidly mounted on a vehicle that is capable of turning around its own vertical axis, it seemed expedient to provide for an independent panning capability. Thus, the TV-range finder combination is mounted on a yoke that can be rotated by a vertically-mounted stepping motor. The yoke accommodates a transverse, horizontal axis, around which the TV camera can be tilted. The tilt drive train incorporates a worm drive and another stepping motor. The worm drive is necessary to cope with the excessive tipping moments originating from a revised version of the range finder. When the stepping motor is not in operation, the worm drive provides a self-locking feature as an added bonus. In the pan mode, limit switches and stops are provided as well as an electromagnetic detent, acting on a 200-tooth gear, mounted on the shaft of a 200-step/revolution stepping motor. The yoke was designed for these functions only. The shaft of the pan motor is coaxially mounted with the vertical centerline of the vehicle; that is, if equal and opposite commands are given to the driven wheels, the location of the pan motor shaft does not change. The TV camera is located in such a fashion that the photosensitive surface of its vidicon tube is exactly at the intersection of the vertical pan axis and the tilt axis. Turning the vehicle about its vertical axis, panning the camera, and tilting it, does not affect the location of the vidicon surface, only its direction.

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

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

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

5. Tactile Sensors

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

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

**From [3], pages 40-45.*

Appendix B

Some Current Techniques For Scene Analysis

Appendix B

Some Current Techniques For Scene Analysis

For completeness, we reprint below an SRI AI Center Technical Note by Richard Duda [28] that describes some of the vision routines used by Shakey.

Some Current Techniques for Scene Analysis
by
Richard O. Duda

I. Introduction

The purpose of the visual system is to provide the automaton with important information about its environment, information about the location and identity of walls, doorways, and various objects of interest. By adding new information to the model, the visual system gives the automaton a more complete and accurate representation of its world. The role of vision is not independent of the state of the model. If the automaton has entered a previously unexplored area, the visual scene must be analyzed to add information about the new part of the environment to the model. In this situation, the model can provide so little assistance that it is ofter not referenced at all. On the other hand, if the automaton is in a thoroughly known area, the role of vision changes to one of providing visual feedback to correct small errors and verify that nothing unexpected has happened. In this situation, the model plays a much more important role in assisting and actually guiding the analysis.

Until recently our attention has been directed primarily at the general scene-analysis problem. Every picture was viewed as a totally new scene exposing a completely unknown area. More recently we have addressed the problem of using a complete, prespecified map of the floor area to update the automaton's position and help in tasks such as going through a doorway. Another use of this kind of visual feedback would be the monitoring of objects being pushed.

In trying to solve these problems, we have tended to take one or the other of two extreme approaches. Either we tried to develop general methods that can cope with any possible situation in the automaton's world, or we tried to exploit rather special facts that allow an efficient special-purpose solution. The first approach involves the more interesting problems in artificial intelligence, but it provides more capabilities than are needed in many situations, and provides them at the cost of relatively long computation times. The second approach provides fast and effective solutions when certain (usually implicit) preconditions are satisfied, though it can fail badly if these conditions are not met. Eventually, of course, some combination of these two approaches will be needed, since the automaton actually operates in a partially known world, rather than one that is completely unknown or completely known. However, we have decided to concentrate on these two extreme situations before addressing the intermediate case. The remainder of this note describes the current status of our work in these areas.*

II. Region Analysis

A. The Merging Procedure

Our work in general scene analysis is based on dividing the picture into regions representing walls, floors, faces of objects, etc. The basic approach has been described in detail elsewhere [16], and only a brief summary will be given here. The procedure begins by partitioning the digitized image into elementary regions of constant brightness. This usually produces many small, irregularly shaped regions that are fragments of more meaningful regions. Two heuristics are used to merge these smaller regions together. Both of these heuristics operate on the basis of fairly local information, the difference in brightness along the common boundary between two neighboring regions. The heuristics are not infallible; they can merge regions that should have been kept distinct, and they can fail to merge regions that should been merged. However, they reduce the picture to a small number of large regions corresponding to major parts of the picture, together with a larger number of very small regions that can usually be ignored.

The effect of applying these heuristics is best described through the use of examples. Figure B-1 shows television monitor views of three typical corridor scenes. Figure B-2

---

*Our earlier work in scene analysis is described in [7]. Additional information on more recent work is contained in [8], [16], [29], and [30].

shows the results of applying the merging heuristics to digitized versions of these pictures. The boundaries of the regions in these pictures are directed contours, and can be traced using the correspondences shown in Table B-1. Generally speaking, important regions can be separated from unimportant regions purely on the basis of size. Figure B-2a, for example, contains four large, important regions. Three of them are directly meaningful (the door, the wall to the right, and the baseboard), and the fourth is the union of two important regions (the floor and the wall to the left). An inspection of Figure B-2b shows similar results. Figure B-2c shows the result of applying the technique to a complicated scene; while some useful information can be obtained, the resolution available severely limits the usefulness of the results.

Our only complete scene-analysis program is oriented toward identifying boxes and wedges, objects with triangular or rectangular faces, in a simple room environment [16]. For this task, we begin by fitting the boundaries of the major regions by straight lines. Regions are identified as being part of the floor, walls, baseboards, and faces of objects by such properties as shape. brightness, and position in the picture. Objects are identified by grouping neighboring faces satisfying some of the simpler criteria used by Guzman [31]. In the process, certain errors caused by incorrect merging are detected and corrected. We have yet to complete a similar analysis program for the conditions encountered in corridor scenes. However, we have investigated the problem of obtaining a scene description that is internally consistent; the next section describes the analysis approach for this problem.

B. A Procedure for Scene Analysis

If we assume temporarily that the merging heuristics have succeeded in the sense that all of the large regions are meaningful areas, then the only basic problem remaining is the proper identification of each region. Examination of the corridor pictures indicates the need to be able to identify a number of different region types, including the following:

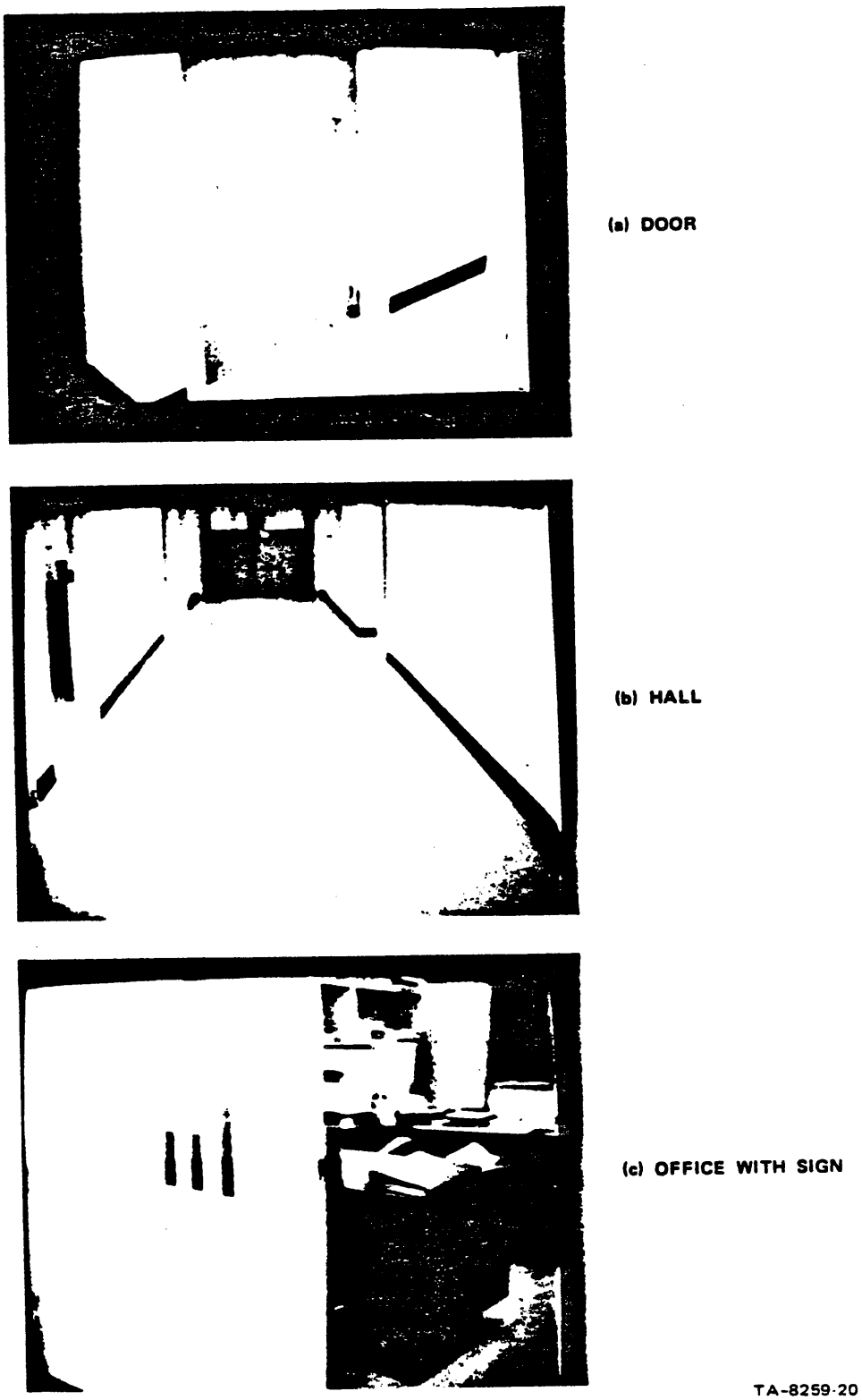


Figure 1: THREE CORRIDOR SCENES

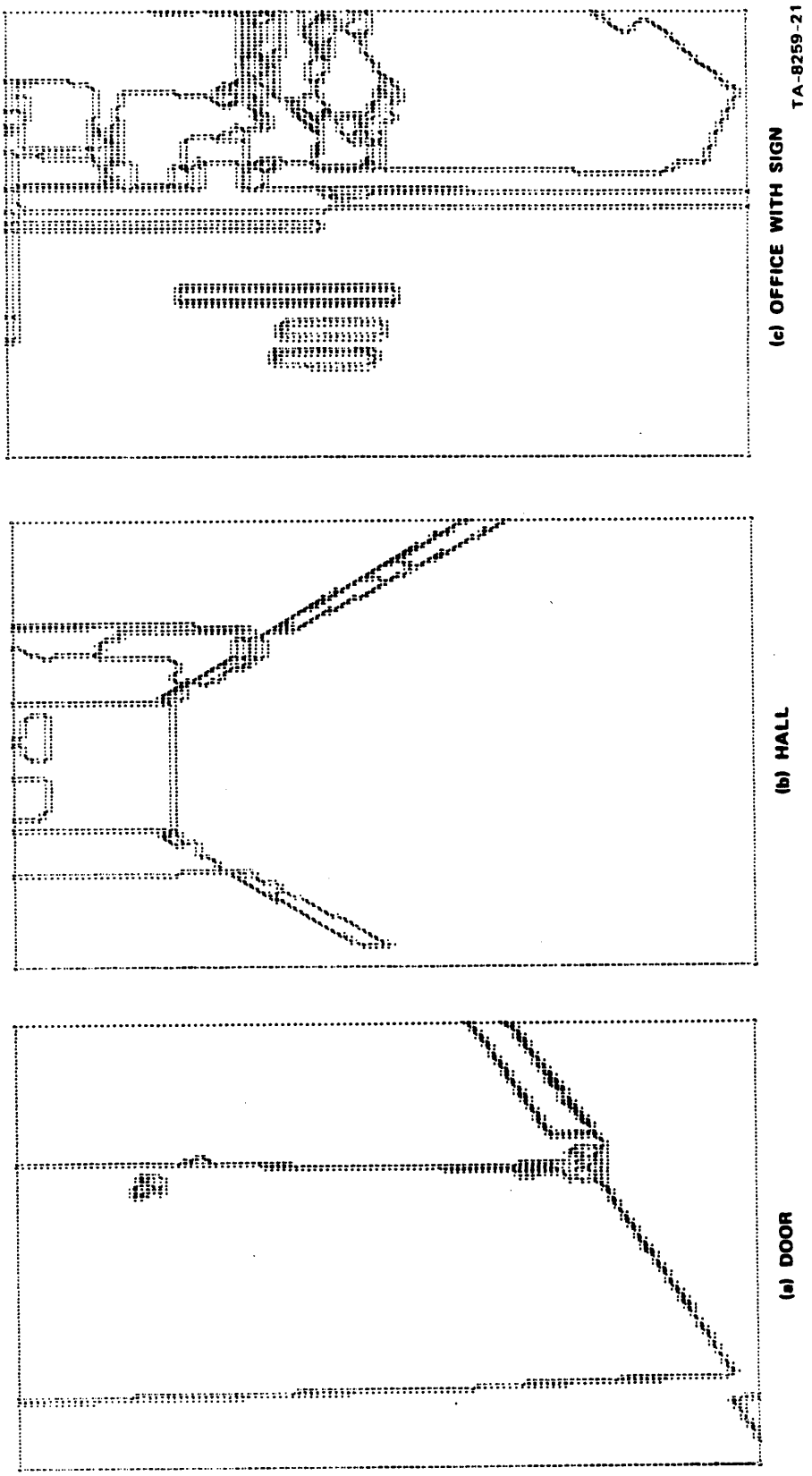


Figure 2: RESULTS OF MERGING HEURISTICS

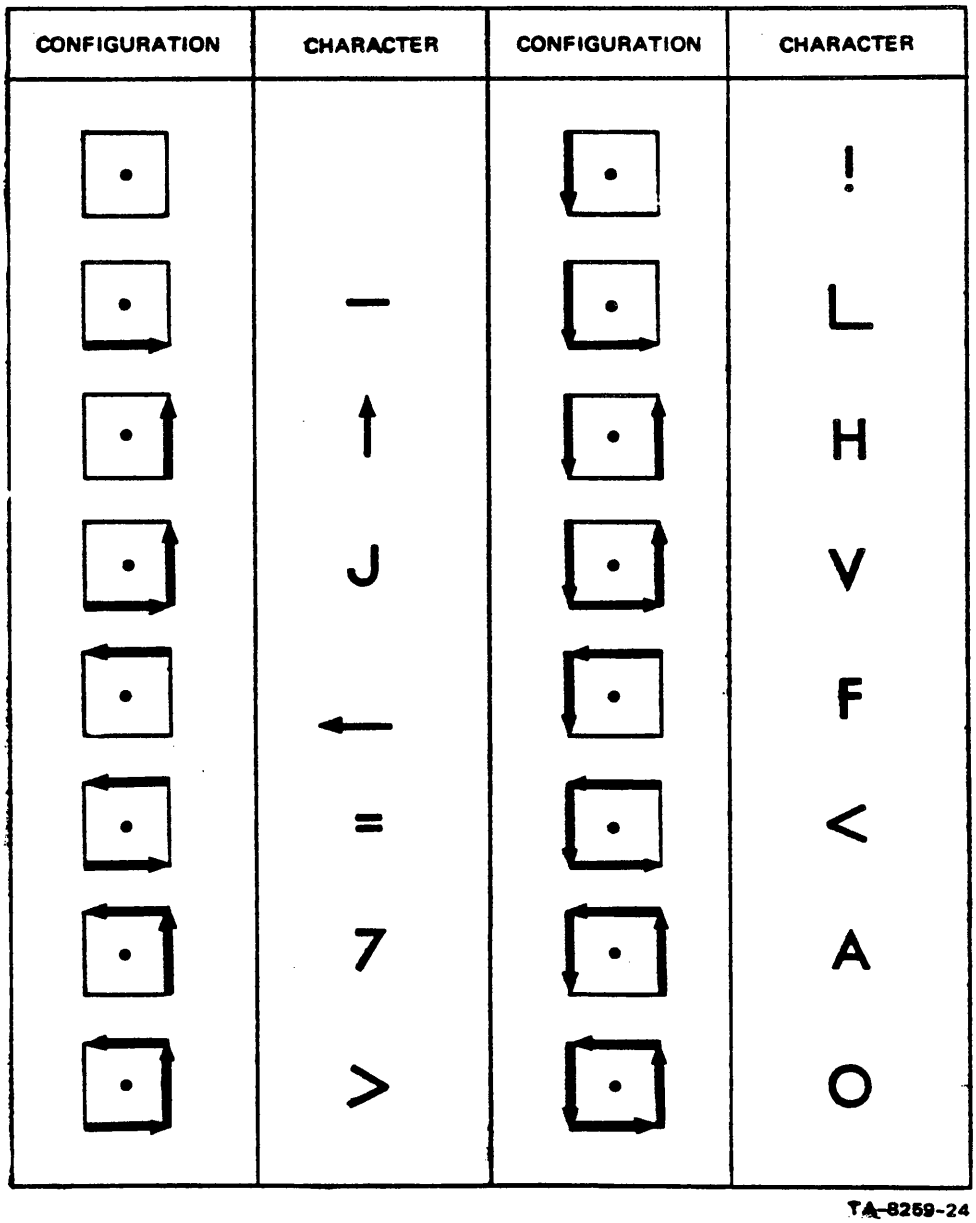


Table 1: CORRESPONDENCE BETWEEN BOUNDARY SEGMENT CONFIGURATIONS AND CHARACTERS USED IN PRINTOUT

(1) Floor
(2) Wall
(3) Door
(4) Door jamb
(5) Object face
(6) Baseboard
(7) Baseboard reflection
(8) Sign*
(9) Window
(10) Clock
(11) Doorknob
(12) Thermostat
(13) Power outlet
(14) Automaton.

Each of these regions has certain properties which tend to characterize it uniquely. For example, the floor region is usually large, bright, and near the bottom of the picture. However, most regions can be identified with greater confidence if the nature of their neighbors is considered as well. Thus, the presence of a baseboard or baseboard reflection at the top of a region almost guarantees that the region is the floor; conversely, the presence of wall area immediately above a region guarantees that it can not be a baseboard reflection. If regions are identified without regard to how that choice affects the overall scene description, the chance for error is increased. Moreover, the resulting description can be nonsensical.

Many, though by no means all, of the relations between types of regions relate to neighboring regions. Table B-2 indicates those types of regions that can and cannot be legal neighbors. We can easily add to this further restrictions, such as the fact that the baseboard must have the wall as a neighbor along its top edge. These are some of the important known facts about the general nature of the automaton's environment. The problem is to use facts such as these to aid in the analysis of the scene.

One approach to solving this problem is to use these facts as constraints to eliminate impossible choices. Suppose that each significantly large region in the picture is tentatively classified on the basis of the attributes of that region alone. Suppose further that a score is computed for each region that measures the degree to which it resembles each region type.** For any selection of names for regions, we can define the score for the resulting description as the sum of the individual scores. Then, we can analyze the scene

---

*By "sign" we mean a dark vertical bar on the wall used, as illustrated in Figure B-1c, to identify an office.

**This score might be interpreted as the logarithm of the probability that the given region is of the indicated type.

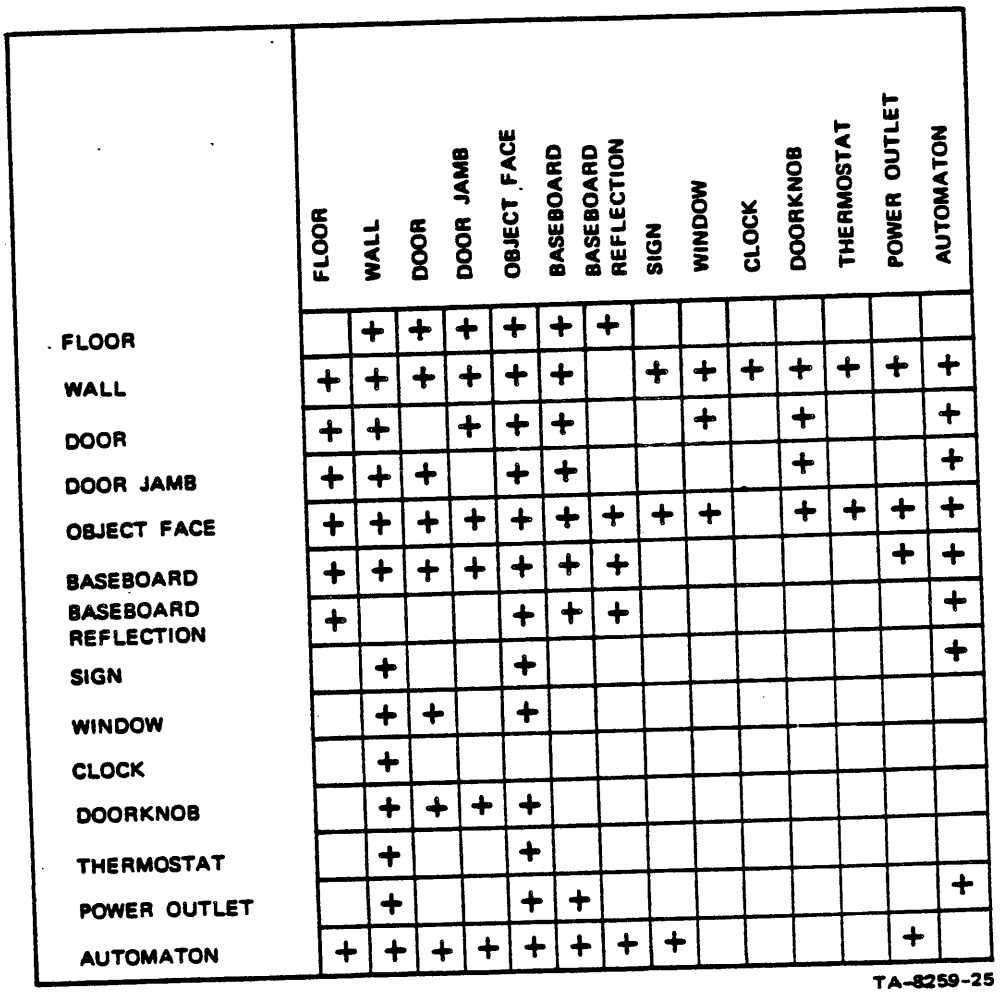


Table 2: REGIONS THAT ARE LEGAL NEIGHBORS

by trying to find highest scoring legal selection of region names. With no loss in generality and some gain in convenience, we can work with the losses incurred by selecting other than the highest scoring choice. In terms of losses, we want the legal description having the smallest overall loss.

This problem is basically a tree-searching problem. The start node of the tree corresponds to the first region selected for naming. The branches emanating from that node correspond to the possible choices of names for that region. A path through the tree corresponds to a unique labeling of the picture. Thus, if there are N possible region names and R regions, there are potentially N^R possible paths through the tree. Each path passes through R+1 nodes from the start node to the terminal node. Every terminal node has a loss value, which is the sum of the losses incurred for the choices along the path to that node. A goal node is a terminal node corresponding to a complete, legal scene description. We seek the goal node with the smallest overall loss.

This is a standard problem in tree searching, and optimum search procedures are known. Assume that some choices have been made for some of the regions so that we have a partially expanded tree. Using the Hart-Nilsson-Raphael terminology [32], some of the terminal nodes of this tree are *open* nodes, candidates for further expansion. Each open node has an associated loss $\hat{g}$, the sum of the losses from the start node to that node. If we assume that there is no reason to believe that zero-loss choices cannot be made from that node on, then the optimal search strategy is to expand that open node having the minimum $\hat{g}$.

To expand a node, we must select a region not previously considered and examine the possible choice for that region, ruling out any choices that are not legal. Different strategies can be used for selecting the next region. It seems advantageous to ask it to be a neighbor of the regions selected previously, since this maximizes the chance of detecting illegalities. In general, we will have several neighbors for candidate successors. Of these, it seems reasonable to select the one having the highest score, under the assumption that the first choice name for this region is most likely to be correct.

After a region has been selected, it is necessary to examine the choices one can make for its name to see which ones are legal. If we limit ourselves to pairwise relations between neighboring regions, we need merely compare each choice with previously made choices on

the path to this point and test each for legality.* The node expanded is removed from the list of open nodes, the resulting new nodes are added, and the process is repeated until the algorithm selects a goal node for further expansion. This is our final result, a legal scene description having the minimum loss.

C. Examples

The following examples serve to illustrate the action of this scene-analysis procedure. Consider first the simple scene shown in Figure B-3. For simplicity, we assume that there are only five types of allowed regions—floor, wall, door, baseboard, and sign. Consider Region 1. On the basis of its brightness, size, vertical right boundary, and possession of a hole, it should receive a high score as wall, and lower scores as floor, door, sign, and baseboard, Region 2 might, perhaps, score highest as a door, and so on. Thus, the following table of scores, although purely imaginary, is not unreasonable. Missing entries correspond to scores too low to be seriously considered.

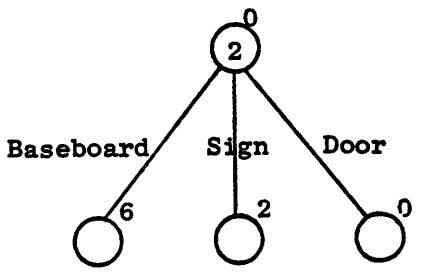
Type / Region	Floor	Wall	Door	Base-board	Sign
1	5	6	2		
2			7	1	5
3	3	3	5		1

*When an illegality is found, that choice is deleted. One can argue that few relations are so strong as to be absolutely illegal, and an alternative approach would be to introduce various additional losses for the different observed relations.

The following table gives equivalent information in terms of the losses associated with each choice.

Region \ Type	Floor	Wall	Door	Base-board	Sign	Max Score
1	1	0	4			6
2			0	6	2	7
3	2	2	0		4	5

Let us use our tree-searching algorithm to obtain the minimum-loss, legal description of this scene. Initially the successor function is unconstrained by neighbor restrictions, and selects Region 2 merely because it has the highest score. At this point, all of the choices for Region 2 are legal, and the tree has three open nodes; the numbers shown next to each node give the loss accumulated in reaching that part of the tree.



The search algorithm requires that the open node having the least loss be expanded next, which corresponds to tentatively calling Region 2 a door. The successor function finds only one neighbor to choose from, Region 1, and considers its alternatives: wall, floor, and door. None of these choices is a legal neighbor surrounding Region 1, and hence all are rejected. Thus, this open node has no successors.

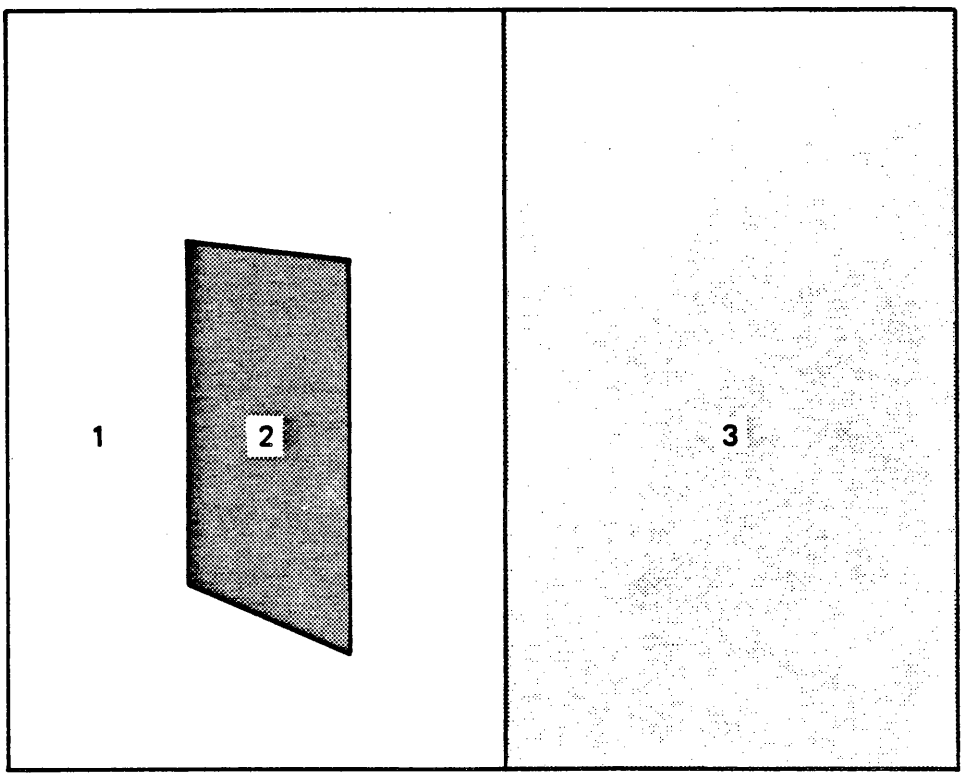
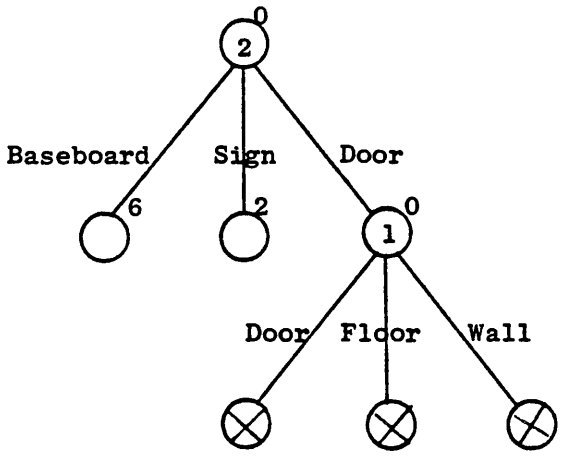
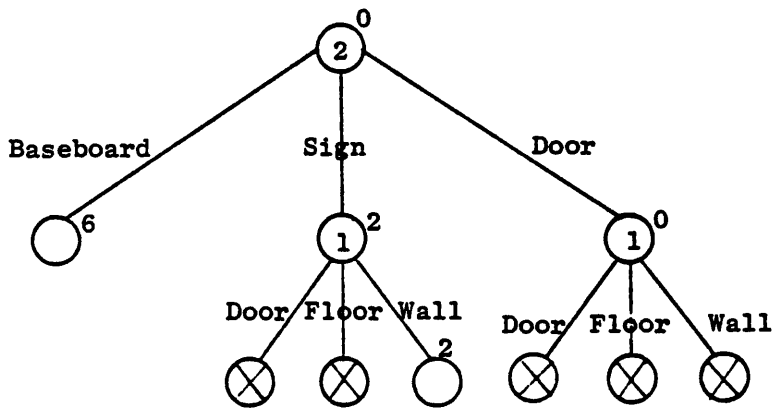


Figure 3: A SIMPLE SCENE



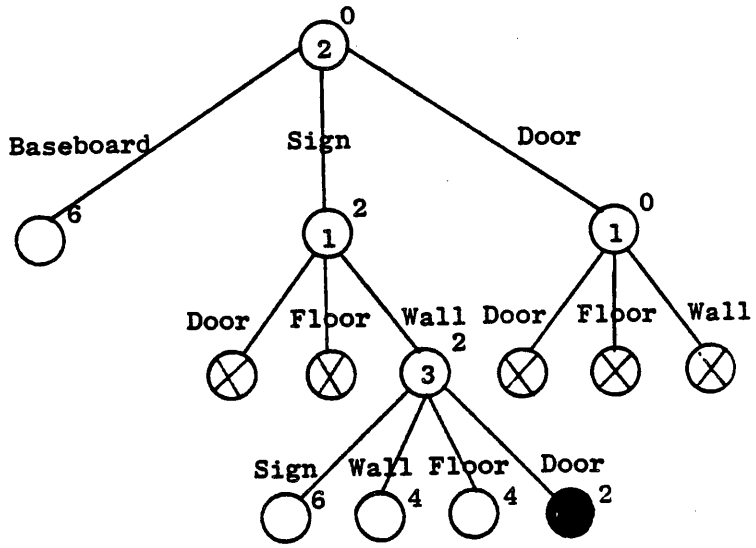
Returning to the choices for open nodes, Region 2 is tentatively called a sign. The successor function again selects Region 1, and this time finds one legal successor, the wall.* The loss associated with this choice is 0, and the overall loss is 2. The list of open nodes still contains two members.



The search algorithm selects the open node with loss 2, and the successor function has only Region 3 to select from. All of the choices for Region 3 are all legal with respect to

*Note that our successor function will always produce a tree with R+1 levels. At any level, the same region will always be selected by the successor function. The actual successors, however, will be limited by the legality requirement.

calling Region 2 a sign and Region 1 a wall. The least loss results from calling Region 3 a door, and the scene analysis is completed.



A somewhat more realistic example involving 10 regions and 14 region types is illustrated in Figure B-4. Table B-3 gives the hypothetical scores. Based on these scores alone, half of the regions would be incorrectly identified. Figure B-5 shows the tree produced by the search algorithm. The development of this tree is too complicated to describe in detail. It should be noted, however, that considerable backtracking occurred because a low-scoring third choice was needed for Region 8, the doorknob. Whether or not this can be circumvented without causing other problems is not known.

D. Remarks

To date, this procedure has only been used on some hypothetical examples. We have modified a general tree-searching program to adapt it to some special characteristics of this problem. However, we have not started the important task of writing programs to measure characteristics of regions and to use these characteristics to produce recognition scores.

In addition, we have not implemented any legality conditions beyond the simple conditions given in Table B-2.

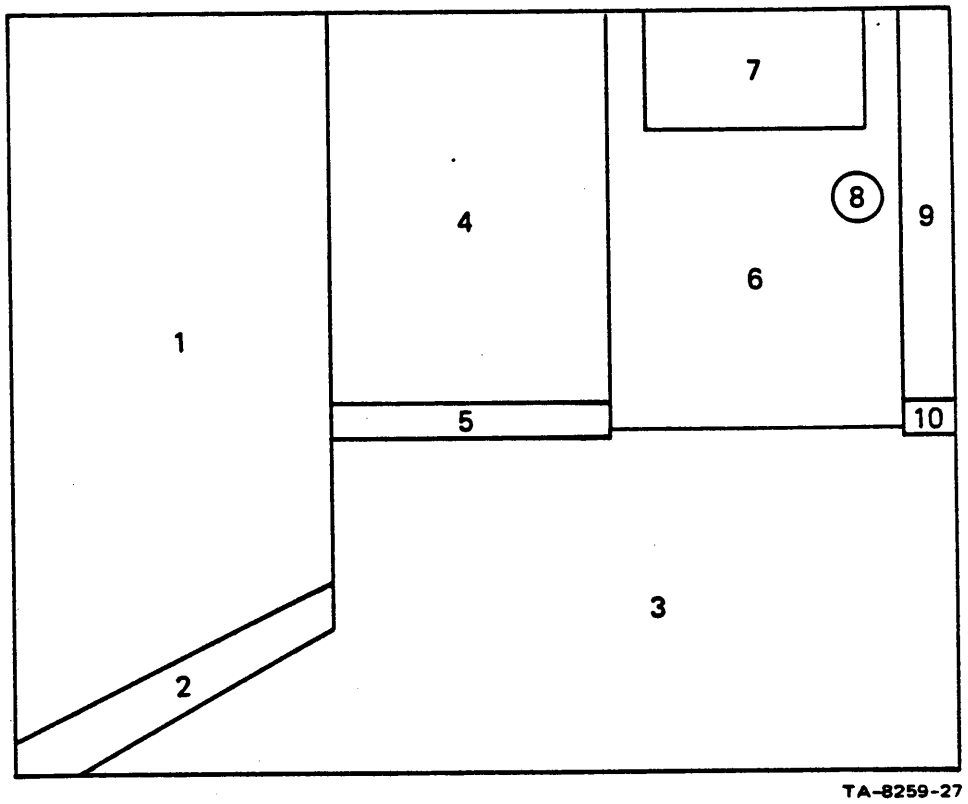


Figure 4: A MORE COMPLICATED SCENE

TYPE	REGION									
	1	2	3	4	5	6	7	8	9	10
FLOOR	1		11			2				
WALL	7		3	5		5			4	
DOOR	3			6		6			3	
DOOR JAMB									6	
OBJECT FACE							6			
BASEBOARD		5			9					3
BASEBOARD REFLECTION		7			5					
SIGN		1								6
WINDOW	1			2			8			
CLOCK								1		
DOORKNOB								2		
THERMOSTAT								6		
POWER OUTLET								3		4
AUTOMATON										

TA-8259-29

Table 3: HYPOTHETICAL REGION SCORES

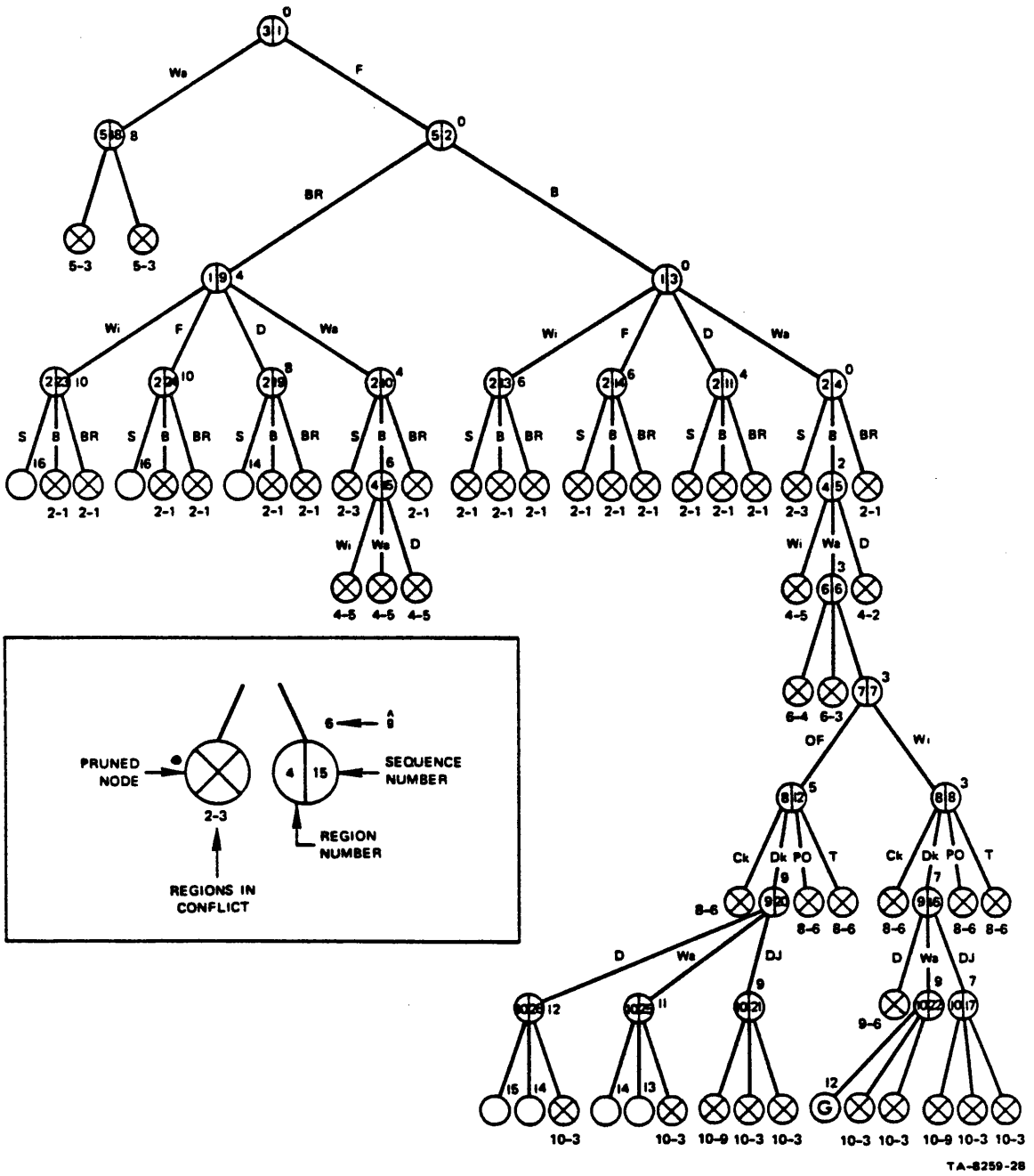


Figure 5: THE ANALYSIS TREE

This approach to scene analysis has several potential advantages. It is not necessary to identify every region correctly at the outset to obtain a correct analysis, provided that the "syntactic" rules are sufficiently complete. By providing a limit on the allowable loss, a partial scene description can be obtained that may be useful even though incomplete. Perhaps most important, the operations of merging, feature extraction, classification, and analysis are clearly separated, allowing fairly independent modification and improvement. In particular, the general knowledge about the environment can be expressed explicitly as rules for legal scenes, and if the environment is changed it is possible to confine the program changes to modifying these rules.

One of the major problems with this approach is the lack of an obvious way to detect erroneous regions, regions that are fragments of or combinations of meaningful regions. We are currently working on this problem, since progress toward its solution is needed before implementation of this system can be begun. Another problem is that it is not clear how specific information contained in the model can be used to guide the analysis. This problem of working in a world that is neither completely known nor completely unknown is one of the major unsolved problems in visual scene analysis.

III. Landmark Identification

When the environment is completely known, the visual system can provide feedback to update the automaton's position and orientation. The x-y location of the automaton and its orientation θ can be determined uniquely from a picture of a known point and line lying in the floor.* Such distinguished points and lines serve as landmarks for the automaton. This section describes our present program that uses concave corners, convex corners, and doorways as landmarks to update position and orientation.

A flowchart outlining the basic operations of this program is shown in Figure B-6. The program begins by selecting a landmark from the model that should be visible from the automaton's present position; if more than one candidate exists, one is selected on the basis of range and the amount of panning of the camera required.* The camera is then panned and tilted the amount needed to bring the landmark into the center of the field of

*If no landmark is in view, a suitable message is returned together with a suggested vantage point from which a landmark can be seen. This is one of several "error" returns that can be obtained from the program. The program can also be asked to select a specific landmark, or a landmark different from the ones previously selected.

view, and a picture is taken. The baseboard-tracking routine described previously [8] is used to find the segments of baseboard in the picture and to fit them with long straight lines.

Exactly what happens next depends on the landmark type. For a door, the long line nearest the center of the picture is selected, and the true image of the landmark is assumed to be the endpoint of the baseboard segment on that line and nearest the center of the picture. An additional check is made to see that the gap from that point to the next segment is long enough to be a passageway. A convex corner viewed from an angle such that only one side is visible is treated as if it were a door. Otherwise, the intersection of long lines nearest the center of the picture is assumed to be the true image of the landmark, and a check is made to see that the baseboard segments near this point have the right geometrical configuration. The location of the landmark in the picture gives the information needed to compute corrections for the automaton's position and orientation.

The operation of this program is illustrated in Figure B-7. In this experiment, the automaton was approximately 7.5 feet away from a wall along which there were four landmarks, both sides of a doorway, a convex corner, and a concave corner. The pictures in Figure B-7. show how closely the panning and tilting brought the landmarks to the center of the pictures. For scenes as clear as these, the program operates very reliably. Presently, we can use this routine to locate the robot with an accuracy of between 5 percent and 10 percent of the range, and to fix its orientation to within 5 degrees. Since the errors are random, the accuracy can be improved further by sighting a second landmark. Further increases in accuracy, if needed, will have to be obtained by improving the tilt and pan mechanism for the camera.*

From [28], pages 1-24

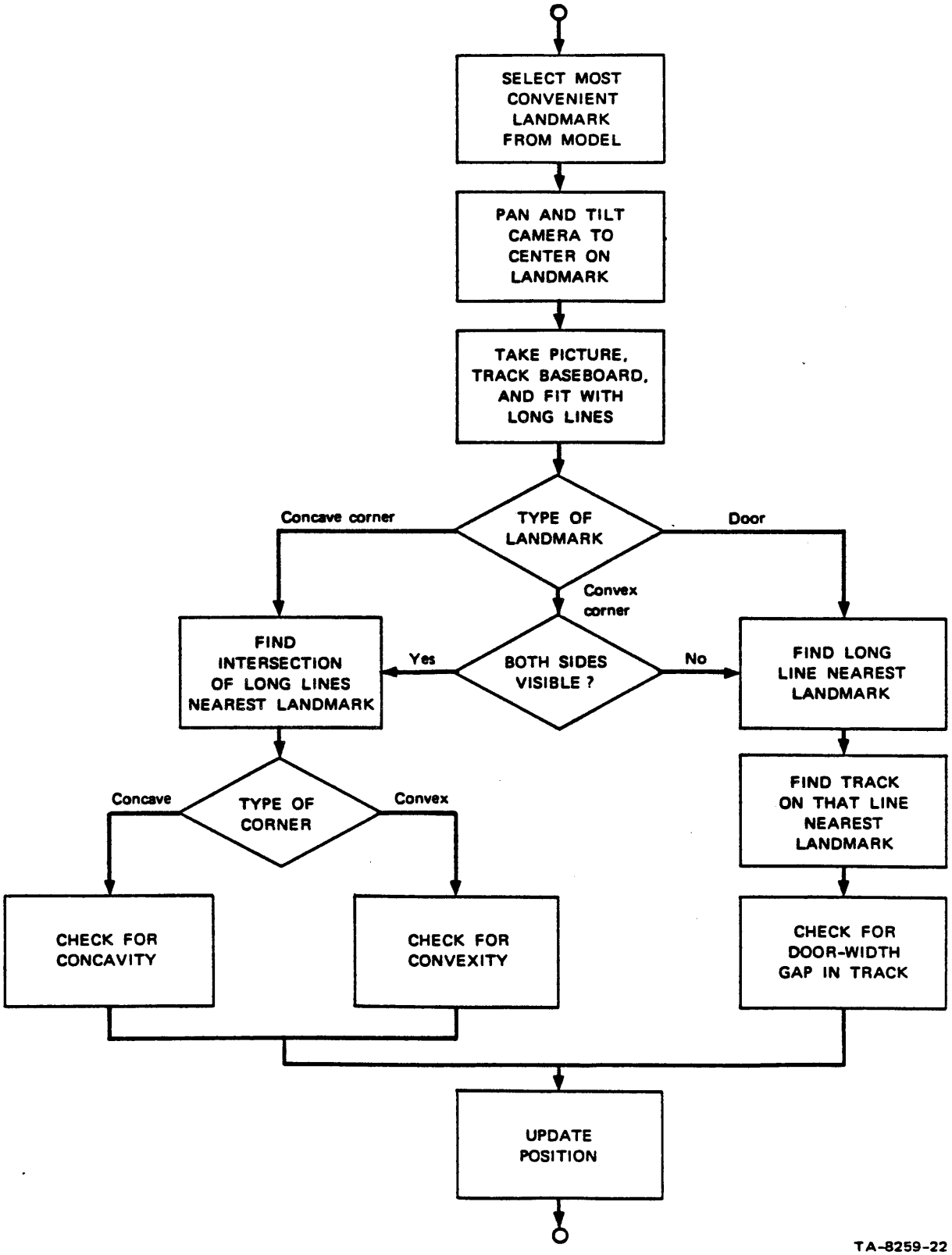


Figure 6: BASIC FLOWCHART FOR LANDMARK PROGRAM

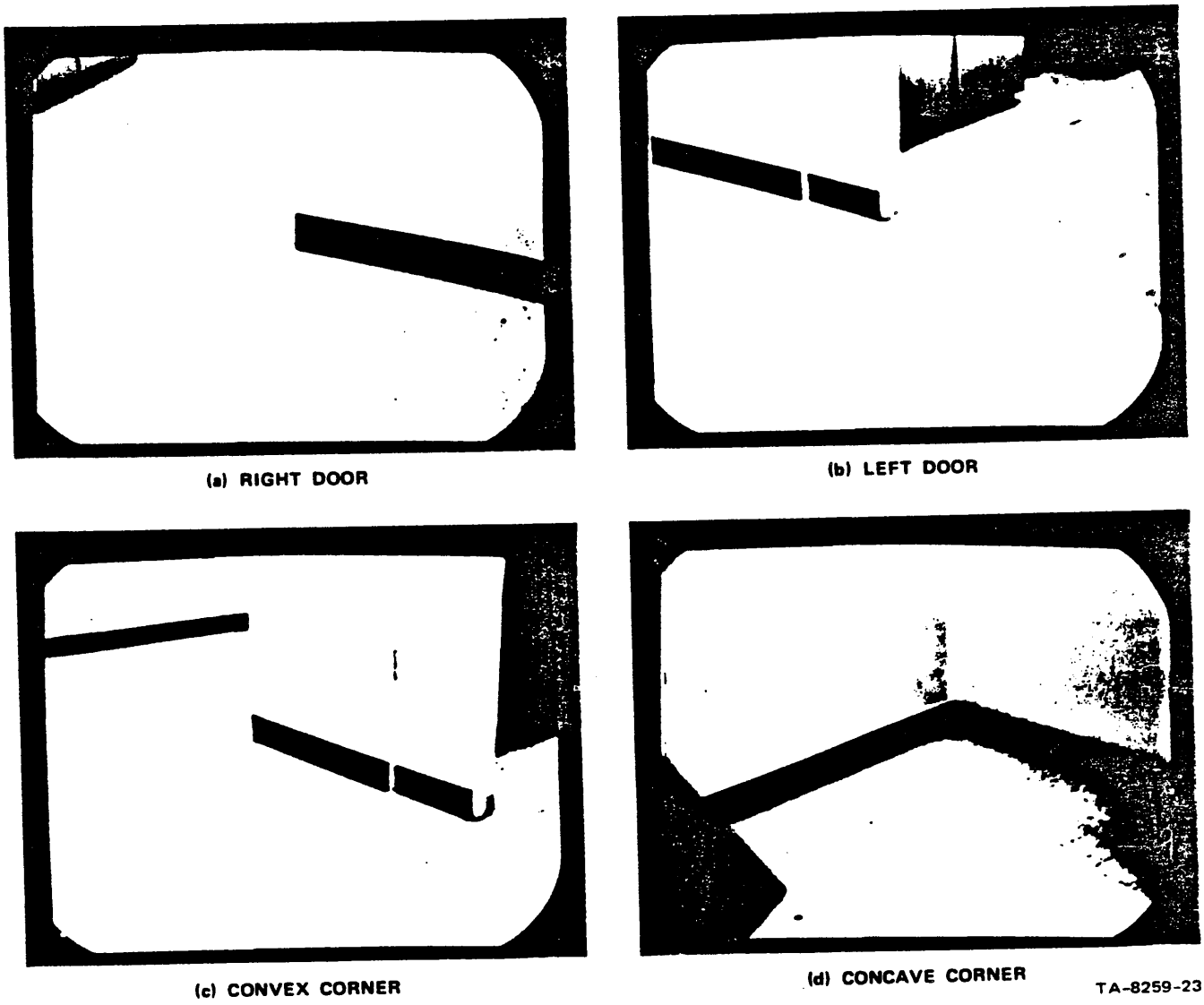


Figure 7: LANDMARKS

REFERENCES

[1] Rosen, Charles A., and Nils J. Nilsson, eds.
Application of Intelligent Automata to Reconnaissance, First Interim Report, prepared for Rome Air Development Center, Griffiss Air Force Base, New York, under contract AF 30(602)-4147, SRI Project 5953, Artificial Intelligence Center, SRI International, Menlo Park, California (November 1966). NTIS access number 80-817 189.

[2] Rosen, Charles A., and Nils J. Nilsson, eds.
Application of Intelligent Automata to Reconnaissance, Second Interim Report, prepared for Rome Air Development Center, Griffiss Air Force Base, New York, under contract AF 30(602)-4147, SRI Project 5953, Artificial Intelligence Center, SRI International, Menlo Park, California (March 1967). NTIS access number 80-820 989.

[3] Rosen, Charles A., and Nils J. Nilsson, eds.
Application of Intelligent Automata to Reconnaissance, Third Interim Report, prepared for Rome Air Development Center, Griffiss Air Force Base, New York, under contract AF 30(602)-4147, SRI Project 5953, December 1967. Artificial Intelligence Center, SRI International, Menlo Park, California (December 1967). NTIS access number 80-827 938.

[4] Nilsson, N., B. Raphael, and S. Wahlstrom.
Application of Intelligent Automata to Reconnaissance, Fourth Interim Report, prepared for Rome Air Development Center, Griffiss Air Force Base, New York, under contract AF 30(602)-4147, SRI Project 5953, Artificial Intelligence Center, SRI International, Menlo Park, California (May 1968). NTIS access number 80-841 509.

[5] Nilsson, N. J., C. A. Rosen, B. Raphael, G. Forsen, L. Chaitin, and S. Wahlstrom.
Application of Intelligent Automata to Reconnaissance, Final Report, prepared for Rome Air Development Center, Griffiss Air Force Base, New York, under contract AF 30(602)-4147, SRI Project 5953, Artificial Intelligence Center, SRI International, Menlo Park, California (December 1968). NTIS access number 80-849 872.

[6] Nilsson, Nils J.
Research on Intelligent Automata, First Interim Report, prepared for Rome Air Development Center, Griffiss Air Force Base, New York, under contract F30602-69-C-0056, (ARPA Order No. 1058, Amendment 1), SRI Project 7494, Artificial Intelligence Center, SRI International, Menlo Park, California (February 1969). NTIS access number AD-A140279

[7] Coles, L. S., R. O. Duda, T. D. Garvey, J. H. Munson, B. Raphael, C. A. Rosen, and R. A. Yates.
Application of Intelligent Automata to Reconnaissance, Final Report, prepared for Rome Air Development Center, Griffiss Air Force Base, New York, under contract F30602-69-C-0056, (ARPA Order No. 1058, Amendment 1), SRI Project 7494, Artificial Intelligence Center, SRI International, Menlo Park, California (November 1969). NTIS access number 80-86A 871.

[8] Chaitin, L. J., R. O. Duda, P. A. Johanson, B. Raphael, C. A. Rosen, and R. A. Yates.
Research and Applications - Artificial Intelligence, Interim Scientific Report, prepared for the National Aeronautics and Space Administration, 600 Independence Avenue, S.W., Washington, D.C., under contract NAS1-2221, (ARPA Order No. 1058, Amendment 1), SRI Project 8259, Artificial Intelligence Center, SRI International, Menlo Park, California (April 1970). NTIS access number N73-25173.

[9] Raphael, Bertram.
Research and Applications - Artificial Intelligence, Final Report, prepared for the National Aeronautics and Space Administration, 600 Independence Avenue, S.W., Washington, D.C., under contract NAS1-2221, (ARPA Order No. 1058, Amendment 1), SRI Project 8259, Artificial Intelligence Center, SRI International, Menlo Park, California (November 1970). NTIS access number N73-72140.

[10] Raphael, B., L. J. Chaitin, R. O. Duda, R. E. Fikes, P. E. Hart, and N. J. Nilsson.
Research and Applications - Artificial Intelligence, Semiannual Progress Report, prepared for the National Aeronautics and Space Administration, 600 Independence Avenue, S.W., Washington, D.C., under contract NASW-2164, SRI Project 8973, Artificial Intelligence Center, SRI International, Menlo Park, California (April 1971). NTIS access number N73-22558.

[11] Raphael, B., R. O. Duda, R. E. Fikes, P. E. Hart, N. J. Nilsson, P. W. Thorndyke and B. M. Wilber. *Research and Applications - Artificial Intelligence*, Final Report, prepared for the National Aeronautics and Space Administration, 600 Independence Avenue, S.W., Washington, D.C., under contract NASW-2164, SRI Project 8973, Artificial Intelligence Center, SRI International, Menlo Park, California (December 1971). NTIS access number N73-23279.

[12] Hart, P. E., R. E. Fikes, T. D. Garvey, N. J. Nilsson, D. Nitzan, J. M. Tenenbaum, and B. M. Wilber. *Artificial Intelligence - Research and Applications*, Annual Technical Report, prepared for the Advanced Research Projects Agency, Arlington, Virginia, under contract DAHCO4-72-C-0008, (ARPA Order No. 1943), SRI Project 1530, Artificial Intelligence Center, SRI International, Menlo Park, California (December 1972). NTIS access number AD7 56970.

[13] Munson, John H. *A LISP-FORTRAN-MACRO Interface for the PDP-10 Computer*, Technical Note 16, Artificial Intelligence Center, SRI International, Menlo Park, California (November 1969).

[14] Green, C. "Application of Theorem Proving to Problem Solving," in *Proceedings of 1st International Joint Conference on Artificial Intelligence*, Washington, D. C., Walker, Donald E., and Lewis M. Norton (eds.), pp. 219-239 (1969). Also appears in *Readings in Artificial Intelligence*, Webber, Bonnie Lynn, and Nils J. Nilsson, eds., pp. 202-222, Tioga Publishing Company, Palo Alto, California (1981).

[15] Garvey, T. D., and R. E. Kling. *User's Guide to QA3.5 Question-Answering System*, Technical Note 15, Artificial Intelligence Center, SRI International, Menlo Park, California (December 1969).

[16] Brice, C. R., and C. L. Fennema. *Scene Analysis Using Regions*, Technical Note 17, Artificial Intelligence Center, SRI International, Menlo Park, California (April 1970). Also appears in *Artificial Intelligence*, 1(3), pp. 205-226 (Fall, 1970).

[17] Duda, R. O., and P. E. Hart.
Pattern Recognition and Scene Analysis, New York: John Wiley and Sons (1973).

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

[19] Nilsson, Nils J.
Principles of Artificial Intelligence, Tioga Publishing Company, Palo Alto, California (1980).

[20] Sacerdoti, E. D.
A Structure for Plans and Behavior, New York: Elsevier (1977).

[21] Wilkins, D.
"Domain-independent Planning: Representation and Plan Generation," *Artificial Intelligence* 22, Number 3, pp. 269-301 (April 1984).

[22] Fikes, R. E., P. E. Hart, and N. J. Nilsson.
"Learning and Executing Generalized Robot Plans," Technical Note 70, Artificial Intelligence Center, SRI International, Menlo Park, California (July 1972). Also appears in *Artificial Intelligence*, 3(4), pp. 251-288 (1972).

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

[24] Fikes, R. E.
Monitored Execution of Robot Plans Produced by STRIPS, Technical Note 55, Artificial Intelligence Center, SRI International, Menlo Park, California (April 1971). Also appeared in *Proceedings IFIP Congress 71*, Ljubljana, Yugoslavia (August 23-28, 1971).

[25] Film, *SHAKEY: Experiments in Robot Planning and Learning*, Artificial Intelligence Center, SRI International, Menlo Park, California (1972).

[26] Coles, L. Stephen.
"Talking with a Robot in English," *Proceedings of the International Joint Conference on Artificial Intelligence*, Washington, D. C., Walker, Donald E. and Lewis M. Norton (eds.), pp. 587-596 (1969).

[27] Fredrickson, T. R.
"Closed Loop Stepping Motor Application," *Third International Federation of Automatic Control Congress*, London (June 20-25, 1966).

[28] Duda, R. O.
Some Current Techniques for Scene Analysis, Technical Note 46, Artificial Intelligence Center, SRI International, Menlo Park, California (October 1970).

[29] Duda, R. O., and P. E. Hart.
Experiments in Scene Analysis, Technical Note 20, Artificial Intelligence Center, SRI International, Menlo Park, California (January 1970).

[30] Duda, R. O., and P. E. Hart.
A Generalized Hough Transformation for Detecting Lines in Pictures, Technical Note 36, Artificial Intelligence Center, SRI International, Menlo Park, California (August 1970).

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

[32] Hart, P. E., N. J. Nilsson, and B. Raphael.
"A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans, Sys. Sci. Cyb.*, Vol. SSC-4, pp. 100-107 (July 1968).