NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Memorandum 33-568

Phase 1 Report on a Cognitive Operating System (COGNOSYS) for JPL's Robot

F. P. Mathur

(NASA-CR-128346) A COGNITIVE OPERATING SYSTEM (COGNOSYS) FOR JPL'S ROBOT, PHASE 1 REPORT F.P. Mathur (Jet Propulsion Lab.) 15 Sep. 1972 35 p CSCL 09B G3/08 N72-33192 Unclas 44065

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

September 15, 1972

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. 33-568	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle PHASE 1 REPORT ON A COGNITIVE OPERATING SYSTEM (COGNOSYS) FOR JPL'S ROBOT		5. Report Date September 15, 1972 6. Performing Organization Code
7. Author(s) F. P. Mathur		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91103		10. Work Unit No. 11. Contract or Grant No. NAS 7-100 13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

The most important software requirement for any robot development is the COGNitive Operating SYStem (COGNOSYS). This report describes the Stanford University Artificial Intelligence Laboratory's Hand/Eye software system from the point of view of developing a cognitive operating system for JPL's Robot. In this, the Phase I of the JPL Robot COGNOSYS task the installation of a SAIL compiler and a FAIL assembler on Caltech's PDP-10 have been accomplished and guidelines have been prepared for the implementation of a Stanford University type Hand/Eye software system on JPL-Caltech's computing facility. The alternatives offered by using RAND-USC's PDP-10 Tenex operating system are also considered.

17. Key Words (Selected by Author(s)) Computer Applications and Equipment Planetary Exploration, Advanced Artificial Intelligence	18. Distribution Statement Unclassified -- Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	34	

HOW TO FILL OUT THE TECHNICAL REPORT STANDARD TITLE PAGE

Make items 1, 4, 5, 9, 12, and 13 agree with the corresponding information on the report cover. Use all capital letters for title (item 4). Leave items 2, 6, and 14 blank. Complete the remaining items as follows:

3. Recipient's Catalog No. Reserved for use by report recipients.
7. Author(s). Include corresponding information from the report cover. In addition, list the affiliation of an author if it differs from that of the performing organization.
8. Performing Organization Report No. Insert if performing organization wishes to assign this number.
10. Work Unit No. Use the agency-wide code (for example, 923-50-10-06-72), which uniquely identifies the work unit under which the work was authorized. Non-NASA performing organizations will leave this blank.
11. Insert the number of the contract or grant under which the report was prepared.
15. Supplementary Notes. Enter information not included elsewhere but useful, such as: Prepared in cooperation with... Translation of (or by)... Presented at conference of... To be published in...
16. Abstract. Include a brief (not to exceed 200 words) factual summary of the most significant information contained in the report. If possible, the abstract of a classified report should be unclassified. If the report contains a significant bibliography or literature survey, mention it here.
17. Key Words. Insert terms or short phrases selected by the author that identify the principal subjects covered in the report, and that are sufficiently specific and precise to be used for cataloging.
18. Distribution Statement. Enter one of the authorized statements used to denote releasability to the public or a limitation on dissemination for reasons other than security of defense information. Authorized statements are "Unclassified—Unlimited," "U. S. Government and Contractors only," "U. S. Government Agencies only," and "NASA and NASA Contractors only."
19. Security Classification (of report). NOTE: Reports carrying a security classification will require additional markings giving security and downgrading information as specified by the Security Requirements Checklist and the DoD Industrial Security Manual (DoD 5220.22-M).
20. Security Classification (of this page). NOTE: Because this page may be used in preparing announcements, bibliographies, and data banks, it should be unclassified if possible. If a classification is required, indicate separately the classification of the title and the abstract by following these items with either "(U)" for unclassified, or "(C)" or "(S)" as applicable for classified items.
21. No. of Pages. Insert the number of pages.
22. Price. Insert the price set by the Clearinghouse for Federal Scientific and Technical Information or the Government Printing Office, if known.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Technical Memorandum 33-568

Phase 1 Report on a Cognitive Operating System (COGNOSYS) for JPL's Robot

F. P. Mathur

JET PROPULSION LABORATORY
CALIFORNIA INSTITUTE OF TECHNOLOGY
PASADENA, CALIFORNIA

September 15, 1972

Prepared Under Contract No. NAS 7-100
National Aeronautics and Space Administration

PREFACE

The work described in this report was performed by the Astrionics Division of the Jet Propulsion Laboratory.

Preceding page blank

CONTENTS

PRECEDING PAGE BLANK NOT FILMED

FIGURES

ABSTRACT

The most important software requirement for any robot development is the COGNitive Operating SYStem (COGNOSYS). This report describes the Stanford University Artificial Intelligence Laboratory's Hand/Eye software system from the point of view of developing a cognitive operating system for JPL's Robot. In this, the Phase I of the JPL Robot COGNOSYS task the installation of a SAIL compiler and a FAIL assembler on Caltech's PDP-10 have been accomplished and guidelines have been prepared for the implementation of a Stanford University type Hand/Eye software system on JPL-Caltech's computing facility. The alternatives offered by using RAND-USC's PDP-10 Tenex operating system are also considered.

I. INTRODUCTION

The most important software requirement for any robot development is what may be termed the COGnitive Operating SYStem (COGNOSYS). The COGNOSYS is to be distinguished from the operating system of the host computer on which the cognitive operating system is implemented and resides. The JPL robot's sensory-motor functions consist of those corresponding to stereo-TV cameras, range finder, arm(s), and vehicle drive mechanisms. None of the robots either in existence or under current development, to the author's knowledge, have this mix of effectors. The Stanford Research Institute's SHAKEY has no arm. The Stanford University Artificial Intelligence (A. I.) Laboratory's Hand/Eye (HE) system has no mobility. The approach taken by these two centers of robotics in the development of cognitive operating systems are distinctively different from each other. Stanford Research Institute's SHAKEY has a cognitive operating system which is designed around a theorem-prover (the QA3-STRIPS-PLANEX approach) whereas the Stanford University A. I. Laboratory utilizes heuristic strategy program to control the serial/parallel execution of a directory of special-purpose subroutines (jobs, modules), where each subroutine, for example, may be a directive for a specific operation on the robot's subsystem.

This report formulates a methodology for developing a cognitive operating system and describes the Stanford University A. I. Laboratory's HE system from the point of view of developing cognitive operating system for Jet Propulsion Laboratory's robot breadboard. In the Phase I of the COGNOSYS task the installation of a SAIL compiler and a FAIL assembler on Caltech's PDP-10 have been accomplished and guidelines set forth for the implementation of a Stanford-type HE software system on JPL-Caltech's computing facility. The alternatives offered by using RAND-USC's PDP-10 Tenex operating system is also considered.

II. METHODOLOGY

The methodology for the development of JPL robot's cognitive operating system (COGNOSYS) from what was evident at the inception of this project to what has evolved to date may be expressed thus: There was no intention to be restricted to in-house capabilities alone but rather to acquire as much benefit as possible by interacting with nationally known artificial intelligence centers. These benefits would constitute in the awareness of the latest developments relating to cognitive operating systems and in broadening the knowledge base of JPL researchers.

Specifically, these interactions would result in the importation of artificial intelligence specific programming language compilers, assemblers, and utility routines. They would also result in the understanding of COGNOSYSs existing or under development at other centers, in the importation of these cognitive operating systems followed by in-house experimentation with them. This understanding and experimentation (either at JPL or at site of origination) coupled with the JPL specific robot requirements would lead to the modification and extension of these (imported) packages to fit the JPL robot's needs.

Should it be indicated, in the course of the study, that the effort required to modify and extend these packages is not commensurate with the effort required to develop these software packages from functional descriptions and flow charts, then requisite steps would be taken to seek an intermediary balanced approach. Such decisions may come about due to incompatibilities of host machines, time-sharing systems, and availability of compilers. Should the host machine compatibilities be of a marginal nature then intermediary steps, between the extremes of: (1) direct import and translation, and (2) total in-house specified development, are indicated. That is to say, the task will comprise some of those imported packages that are directly utilizable, those that will be utilizable after some modification, other packages that may not thus be utilizable but must be developed and written from basic specifications, and packages which do not exist anywhere, are JPL specific, and hence must be developed here (e.g., those relating to a robot functioning in a Mars environment).

III. OVERVIEW OF HAND/EYE SYSTEM

The HE system consists of a group of jobs which are all constrained to some particular conventions (Fig. 1). These conventions enable communication of data and control information among the jobs. For the purpose of clarity, these separate jobs may also be referred to as modules. Each module represents a logical physical section of the HE system.

All of these modules are run as pseudo-teletype (PTY) jobs under the PDP-10 timesharing system. The user is provided with a teletype (TTY) controller which is responsible for communicating with the various modules in the system. The TTY controller allows commands to be passed to these modules and allows output from the modules to be shown to the user.

The PTY mechanism is used for controlling the modules to accommodate the timesharing system (e.g., logging in, executing system commands, etc.). Since this is not a practical way of communicating large quantities of data, another mechanism has been provided for making data available to all modules and for communicating data between modules. This mechanism makes use of the second segment on the PDP-10. All modules share a common second segment which contains the SAIL routines and global data storage space.

Since the second segment is common to all modules, it may also be used for passing information from one module to another. This information is passed in the form of "messages" which resemble SAIL procedure calls. Messages promote a means for passing data and for requesting executing of so-called "message procedures" in the various modules.

A. Hardware Overview

The HE system's visual input is accomplished by using a commercial TV camera. The camera has a four-lens turret, a four-position color wheel in front of the vidicon, a pan-tilt head, focus, and target voltage all under program control. The arm is powered by small electric motors mounted on it. Each of the joints has a potentiometer mounted on it to provide position feedback. The hand is a two-finger parallel grip device. The TV and arm are connected through analog-to-digital converters to a Digital Equipment

PDP-6 and a PDP-10 computer linked together and sharing 128K of core (which has recently been augmented to the full 256K of core).

All analog-to-digital and digital-to-analog convertors interface with the PDP-6. All I/O devices between the HE and computing system are attached to the PDP-6. The PDP-6, in general, is used for real-time applications such as servoing the arms, changing lenses, changing color filters, pan, tilt, etc.

B. Storage Requirements

The approximate estimate on storage requirements for the assembler, compiler, jobs, global segment, and HE monitor are the following:

FAIL assembler	19 to 42K
SAIL compiler	23 to 50K
HE defined jobs	40K and more
Global models and run time routines	27K
Other storage allocations	3 to 4K
HE monitor	6K

C. Software Overview

The HE system runs under Stanford's PDP-10 timesharing system, which has been modified to enable the HE system to function in a timesharing environment. The HE system is partitioned into many intercommunicating modules. Each module runs as a separate job under the PDP-10 timesharing system. This division alleviates job sizes limitations. It also allows the timesharing scheduler to overlap computation-limited operations like arm servoing. There are, however, two inefficiencies associated with the use of multiple jobs to avoid core overlays; one in the overhead of trapping and routing I/O from all jobs through a single terminal. The second is the difficulty of bringing task-dependent strategies to bear on scheduling decisions.

Most of the HE system is written in SAIL (except for the run-time routines which are mostly in FAIL). SAIL is an ALGOL-like language which contains the LEAP associative processing language. To enable various sections to run asynchronously, and to fit it into core, the system runs as eight separate programs. The PDP-10 has two relocation registers, allowing a program to be in two disjoint segments in core. One of these segments, known as the upper segment, is common to all the programs and contains reentrant subroutines common to all programs. In addition, it contains data which provides a complete global model of the world as it is known to the system at any given time. This model is generated by the lower segment programs and can be interrogated by them. It is predominantly in the form of LEAP associations.

The HE monitor (which resides in the lower segment) is the only program that communicates directly with the operator. It activates PTYs and logs in jobs through them. All characters sent to a PTY by the monitor go to the teletype input buffer of the job attached to the PTY, and any teletype output from a job is available to the monitor. The monitor also contains facilities for directing teletype input to the proper job, outputting teletype output from the jobs to the operator with the job identified, tracing the teletype I/O and message procedure calls for debugging, and setting up and controlling the other jobs. Jobs may also activate a message procedure in the monitor to send commands to it (Figs. 2 and 3).

D. Data Representation: LEAP Triplet Association

An important form of storage of item instances is the association, or triple. Ordered triples of item instances may be written into or retrieved from a special store, the associative store. The method of storage of these triples is designed to facilitate fast and flexible retrieval. A triple is represented by:

$$\text{Attribute} \otimes \text{Object} \equiv \text{Value}$$

where A, O, and V are items or item vars and are mnemonics for attribute, object, and value, respectively.

Examples:

(1) BLOB (X) TABLE [i, j] ≡ blobs known to be in area where TABLE is an item var array (whose indices are X/4, Y/4 where X and Y are in inches and are table coordinates) and where BLOB is the set of connected edges traced by the edge follower (it may be one or more objects).

(2) COLOR (X) CUBE ≡ RED
which reads "color of cube is red."

(3) COLOR (X) ? ≡ RED
which defines the set of all red objects.

E. Strategy or Control Program

The heart of the HE system is the control program. The control program sequences the various tasks, attempts error recovery, generates displays, and has provision for running parts of the system by themselves for debugging. The strategy or control program that exists at Stanford University is a program that enables the HE system to autonomously solve the "Instant Insanity" puzzle (Fig. 4). The puzzle consists of four cubes, each with faces variously selected from four colors: white, blue, red, and green. To solve the puzzle, the blocks must be stacked so that each of the four sides of the resulting tower reveals only one face of each color. Determining the orientation of the cubes in the tower is normally quite difficult for humans. For the computer this is relatively easy. Most of its time and effort is spent in locating and identifying objects, determining the colors of the faces, and, having found the final orientation, deciding what arm motions are required to physically produce the tower.

IV. PSEUDO-TELETYPES

A PTY is an artificial construct within the system to allow users to have and control more than one job at a time. If you do output PTY, it is as if you were sitting at a teletype typing those characters that you outputted. The PTY reads your characters just as a regular teletype does. If you send the character "Login" followed by a carriage return, line feed to a PTY, it will log in a job just as if you had typed that to a teletype. The PTY will then type back the duplexing of what you typed as well as the usual message the system puts out when someone logs in.

The job which initiates a PTY owns it uniquely, and no other job may appropriate that PTY. Using the PTY unused operations (UUOs) one can accomplish from a program anything one can from a command by sending the command to the monitor and performing a PTY UUO with the line number set to zero. That is to say, if you perform a PTY UUO with the line number in ADR set to zero, it is as if the user had typed those characters you outputted. Thus, a job can stop itself by sending control-C to line number zero.

A. Hand/Eye PTY Mechanism Procedures

The program designed to handle PTYs for the HE system (Fig. 5) consists of the following procedures:

DPYCLEAR:	Turn off display frame, put out by job I (3 displays only for now).
DOIT:	Procedure to set and reset flags (used in command decoder).
CORE:	Procedure to determine job size.
STRTST:	Procedure to indicate when string space nearly empty.
TIMOUT:	Procedure to output millisecond time as MIN; SEC; FRACTION.
FORM:	Procedure to format strings.

TRACE: Message procedure tracing functions:

- GETVAL
- GETREAL
- GETSTRING
- GETBITS
- GETARGS

MON-COM: Procedure to send commands to the monitor from the jobs.

SCANLOOP: Procedure to scan the TTY and all the logged in PTYs to see if there is input waiting, and to take appropriate action.

TYPEX: Procedure which types all strings to TTY; it handles suppress and trace processing.

Procedures to help control the PTYs:

HALT: Halts the job ID number in COMJOB.

SEND: Sends strings at the PTY for a job ID number.

SNARF: Waits until a certain character is seen from that PTY.

SNARFMON: Arranges for that PTY to be in monitor mode.

WAITI: Waits for a character from a given PTY and returns it.

COMSCAN: Command scanner. It is called if the scanner loop detected that there was input from the TTY. It checks to see if there is a new job destination and, if so, stores the logical name. If there is a command, the logical name of the destination and the job ID number are stored. If there is no command, the line is typed at the appropriate PTY job.

COMMAND: Command decoder. It is called by COMSCAN if a command is detected. This parses the command, looks it up in the command table, and may then parse arguments to the command. The command name and parsed arguments are stored in ARGS array. Then dispatch is made on the command number for the command. This dispatch is in the form of one big case statement.

V. GLOBAL MODEL

The HE system is composed of several distinct jobs or modules all running independently for the purposes of the time-sharing system. However, these modules will actually be about one common task, are able to communicate with each other. This communication is implemented in two ways: a global data space located in a second segment shared by all the hand/eye modules, and a facility for passing messages between the modules.

All HE modules have access to all the data stored in the global area. The declarations for global data are all included in a declaration tape that precedes the SAIL compilation of each module. This insures that space is allocated such that each separate module knows the same name for a given price of global data (thus avoiding the FORTRAN COMMON problem).

The contents of the global tape are arrived at by agreement and precede each SAIL compilation to be loaded as part of the HE system.

A. Parallel Processing Using Spacewar Mode

Spacewar mode is essentially a parallel process. A job designated in Spacewar mode and started up runs independently from the main job.

One of the important points in a timesharing system is that users' requests for time are scheduled. As a user uses more and more time, his priority goes down and he gets larger and larger time slices. However, completely invisible to the user, his program gets shut off periodically to allow other users to run. This means that no user gets continuous service, but they all get interrupted and shut off periodically. There exists a need for perfectly regular service; e. g., if the SU's hydraulic arm were in operation, a shutdown of any length would cause the arm to wilt. It is for this reason that a mode of operation exists that guarantees perfect (almost) regular service — namely, the Spacewar mode.

When a Spacewar job is initiated, the initiator specifies the time intervals between startups. The Spacewar job will be started from the beginning after that amount of time. While the Spacewar module is active, this job is locked into core and may not be swapped out.

B. Message Procedures and Forward Message Procedures

Message procedures (MPs) provide a mechanism for communicating among the various modules of the HE system. Each of these modules communicate with the common second segment, hence the intra-module communication paths are established in that segment.

Messages are passed back and forth in the second segment. The history of a message may be some subset of the following sequence:

(1) Message is composed.
(2) Message is put in sequence.
(3) Message is "sent."
(4) Wait for completion of the message.
(5) Activate the message (call the procedure).
(6) Acknowledge the processing of the message.
(7) Kill the message.

The capability is needed to send messages that have SAIL-like data associated with them. It is not desired to convert all message data to some symbolic form and (say) write a disk file with that text, but instead to pass data of all types (sets, items, arrays, integers, reals, etc.) in a reasonably efficient manner. At the same time it is desired that programs do not have to explicitly type-check message data or explicitly have to do "get this datum" operations.

A mechanism which meets the above requirements is already in SAIL, namely, actual parameter passing to procedures. A message, then, will consist of a name of a procedure and a parameter list to pass to that procedure for evaluation, together with some bookkeeping information. The user is allowed to specify a symbolic source and a symbolic destination of the message. These names specify the module to be activated (i.e., the recipient of the message), and the source module.

Thus a mechanism is implemented for a user in one module to emit calls to procedures actually located in another module. The matching and passing of formal parameters is handled in much the same way as for ordinary procedures. Of course, the calling module must have declared the names and parameter lists of the procedures it is calling. These declarations will be in the HE definition tape and will look like ordinary procedure declarations, except that the words FORWARD MESSAGE PROCEDURE appear.

A mechanism must be provided in the module in which this procedure is actually located in order to allow this procedure to be evaluated for each message passed to it. It could be arranged that whenever a message specifying the evaluation of some procedure was passed to a module, that module is interrupted and the message request honored. But this is unthinkable, for many reasons. First, the module should control the priorities with which messages are evaluated. Second, it would be objectionable to suspend the module in the midst of a computation which has left an inconsistent view of the world in its data structures.

To rectify this, a module must specifically receive messages, and must request the evaluation of the specified procedure. Briefly, a module may look around in the list of messages in order to locate one destined for itself. It may then request that the message be activated, i.e., evaluate the procedure which is located in the module reading the message and which has the same name as the "procedure name" specified in the message. This evaluation is performed with the arguments as specified in the message. Normally, when the procedure exists, the message is acknowledged (i.e., the calling module may now determine that the message has completed).

C. Tracing

There is a facility for tracing messages passed from one job to another (Fig. 6). This facility is actually handled by the same program which handles the TTY-PTY operations. A trace consists of a type-out at the controlling TTY of the form: "time MESSAGE TRACE: source destination message-procedure-name args" where time is in milliseconds since midnight. Args is a list of argument data for the message procedure. The mechanics of tracing are that there is a global variable in the second segment called TRACING. If it is set non-zero, message tracing is enabled. Every time a message is sent by the message handler, a trace message is first sent to the tracing job. When the tracing message is acknowledged, the original message is finally sent to its prescribed destination. An example of a trace that was conducted on the HE system is included in Appendix C of this report.

VI. UUOs, CALLs, AND CALLIs

The unused op codes from Ø4Ø to Ø77 (in octal) are not used by any instruction and are made use of to communicate with the monitor. These are the UUO codes. An UUO is an instruction which is executed by the system instead of by the computer. These UUOs are used for such functions as to initialize devices, to set up buffer rings, to manipulate files, to make data transfers, to terminate I/O, and to deal with specific I/O devices such as teletypes, magnetic tapes, display units, and DECtapes. Op-codes Ø4Ø through Ø77 and ØØØ trap to absolute location 40, with the central processor in executive mode, and these programmed operators are interpreted by the monitor to perform I/O operations and the functions in the foregoing description.

The previous paragraph described functions of the monitor UUOs. There are also User UUOs, which are op-codes ØØ1 through Ø37, and which allow the user program complete freedom in the use of these programmed operators while not affecting the mode of the central processor.

Op-codes Ø4Ø through Ø77 limit the monitor to $4Ø_8$ operations. The UUØ Ø4Ø, which is the CALL operation, extends this set by specifying the name of the operation by the contents of the location specified by the effective address. This capability provides for indefinite extendability of the monitor operations.

However, the CALL mechanism introduces an overhead cost of a table lookup to the monitor. Thus there is a programmed operator extension of the UUØ Ø47 referred to as CALLI. The CALLI operation eliminates the table lookup of the CALL operation by having the programmer or the assembler to perform the lookup and specify the index to the operation in the effective address of the CALLI AC, N instruction, where N is an index to the operation.

The PDP-10 operating system of the Stanford University A. I. Laboratory recognizes CALLIs up to N = 41 as standard, i. e., these were the standard CALLIs that came with the operating system supplied to them by DEC. These CALLIs (also loosely referred to as UUOs) have been extended by Stanford; i. e., new ones have been defined. In fact, 46 new CALLIs have been defined, bringing the total to 87.

However, in the meantime DEC has not been idle, and in their new versions of their PDP-10 operating system (50 series) 107 CALLIs are defined, i.e., sixty-six new CALLIs have been defined since they supplied their operating system to Stanford. No doubt the impetus to do this may have well come from the ideas developed by Stanford.

Nevertheless, in performing the task of developing an HE-type monitor at JPL by "fitting" the Stanford HE monitor to Caltech's PDP-10, the availability of these new CALLIs is significant. These new CALLIs can now be used to replace many of the Stanford specific ones; e.g., DEC's CALLI AC, 60 has the function of locking jobs in core so that they may not be swapped out, whereas Stanford has a number of SPACEWAR UUOs (see Subsection V.A) that perform functions toward similar objectives.

In summary, although DEC now provides CALLIs that are similar to those developed at Stanford thus making "translation" to Caltech's PDP-10 easier, it should be noted that they are only functionally similar and may not necessarily enable simple direct replacement. This issue will be investigated, in Phase II of this task.

A list of Stanford's standard DEC CALLIs as well as their own defined CALLIs is attached in this report (Appendix B).

A. Summary of Phase I

The two major trends in cognitive operating system design were referred to in the introduction, namely Stanford Research Institute's theorem-prover-based QA3-STRIPS-PLANEX approach and the Stanford University Artificial Intelligence Laboratory's approach, which is to use a heuristic strategy controller of a directory of jobs.

Initially some effort was made to survey theorem-proving techniques and theorem-prover-based question-answering systems. Along these lines the QA 3.5 package developed by Cordell Green and associates at Stanford Research Institute (SRI) was obtained and installed on Caltech's PDP-10. After very little experimentation it was evident that theorem-prover-based deductive systems are indeed very slow. Their strength lies in powerful deductive capability on deep but narrow searches. For broad axiom bases the inference space rapidly gets out of hand, thus reducing speed and requiring large amounts of core storage.

The QA 3.5 package is on Caltech's System directory and is available to anyone with a valid account number to the PDP-10.

Due to the above limitation of theorem-prover-based systems and also due to the broad general requirements for the JPL-Robot's Mars application, along with the consideration that the Stanford University's Shineman arm is being acquired for the JPL-Robot the decision was made to pursue Stanford University A.I. Laboratory's approach. Along this line an effort was initiated to study their system and bring the HE system in-house for experimentation and extension.

Toward this goal a SAIL compiler and a FAIL assembler were installed on Caltech's PDP-10 and are currently being used to gain proficiency in their usage.

The greater part of this report attempts to document the Stanford University A.I. Laboratory's HE system. A summary list of items accomplished in Phase I of this study are:

(1) Investigated theorem-proving techniques.

(2) Investigated question-answering systems.

(3) Acquired SRI's QA 3.5 program and make it operational on Caltech's PDP-10.

(4) Experimented with QA 3.5 at Caltech.

(5) Investigated English language (a subset of the natural language to first-order predicate calculus translators for the purposes of having more convenient front-ends to question-answering systems. Acquired tape of Stephen Cole's ENGROB (English Robot) program from SRI.

(6) Investigated problem-solving programs such as SRI's QA4 and Carl Hewitt's PLANNER at MIT. Obtained a tape of a version of Terry Winograd's implementation called MICROPLANNER.

(7) Investigated PDP-10 Tenex operating system, paging capabilities, fork structure, and communications capabilities.

(8) Investigated Caltech's version 5 PDP-10 operating system.

(9) Initiated dialog with Stanford University A.I. Laboratory personnel.

(10) Formulated methodology for developing a cognitive operating system for JPL Robot.

(11) Acquired documentation on Stanford's HE system based on PDP-10 and PDP-6 computers.

(12) Acquired computer listings of HE monitor, global segment run time routines, and message procedures.

(13) Acquired mag tape of the complete Stanford HE system.

(14) Made listings of the HE system tape at JPL.

(15) Acquired tapes of DECUS's version of SAIL and FAIL.

(16) Made SAIL and FAIL operational on Caltech's PDP-10.

(17) Documented the salient features of Stanford's HE system for the purposes of importation to JPL.

(18) Formulated guidelines for Phase II and estimated magnitude of manpower requirements for the completion of this task.

B. An Estimate of Phase II

During Phase I, the general problem solving area was surveyed for applicability to the development of a cognitive operating system for the JPL-Robot. The emphasis was placed on bringing in-house Stanford University A.I. Laboratory's HE software system. Toward this end the HE system was studied in some detail, and the software infrastructure (SAIL compiler, FAIL assembler, etc.) was established on Caltech's PDP-10.

Along with the acquisition of an understanding of the HE system, Digital Equipment Corporation's latest 5 series version operating system was studied. This revealed that many of the features, such as TTYs, upper segment writability, and special CALLIs which were pioneered at Stanford, have now been incorporated into the Standard 10/50 DEC operating system. Thus the operating system of Caltech's PDP-10 makes available to the user the PTY mechanisms, provides the capability to remove write protection from upper segment under program control, and provides an extended set of CALLIs. These extensions of DEC's capabilities make the implementation of Stanford's HE system at Caltech quite feasible.

Thus, of the primary modules of the HE system, the one that will require the most effort will be in the implementation of the "message procedure" mechanism (which enables jobs to communicate with each other via the global segment).

It is recommended that the transition first be made to the standard 10/50 PDP-10 system. Once that is accomplished, then operation of the system under Tenex 10/50 compatibility mode (either in BBN's Tenex or under Tenex mode of DEC's KI10) should be initiated. The next step should be that of rewriting the system to make use of Tenex's paging features, fork communications, and backtracking capabilities.

The manpower requirements for Phase II of this task, i.e., to have an operational HE-type software system on the PDP-10 in the Booth Computing Center at Caltech is estimated to be between 3 and 6 man-months now that a clear understanding of Stanford's HE system has been gained and the software infrastructure to do the job has been established.

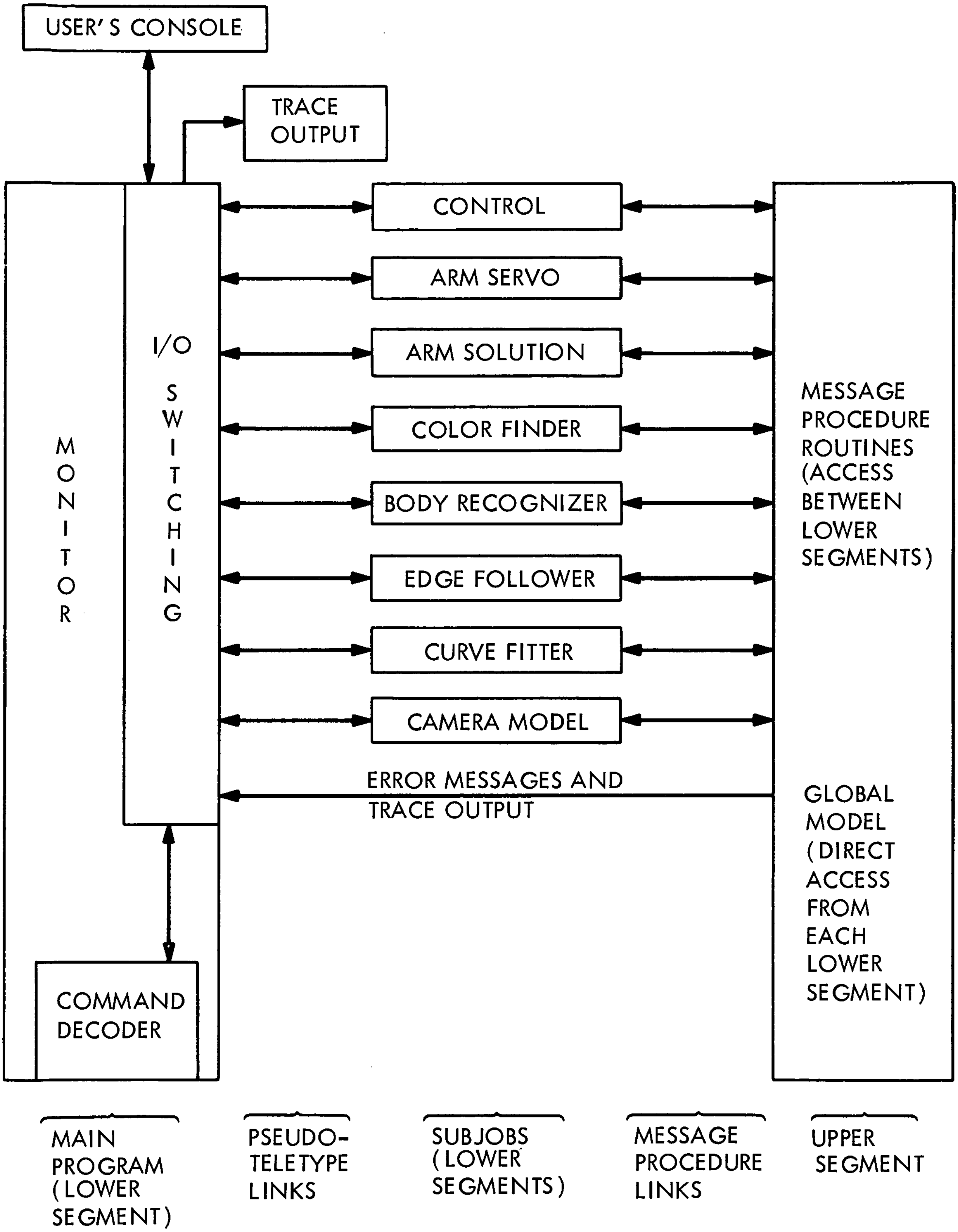

Fig. 1. Overview of the system

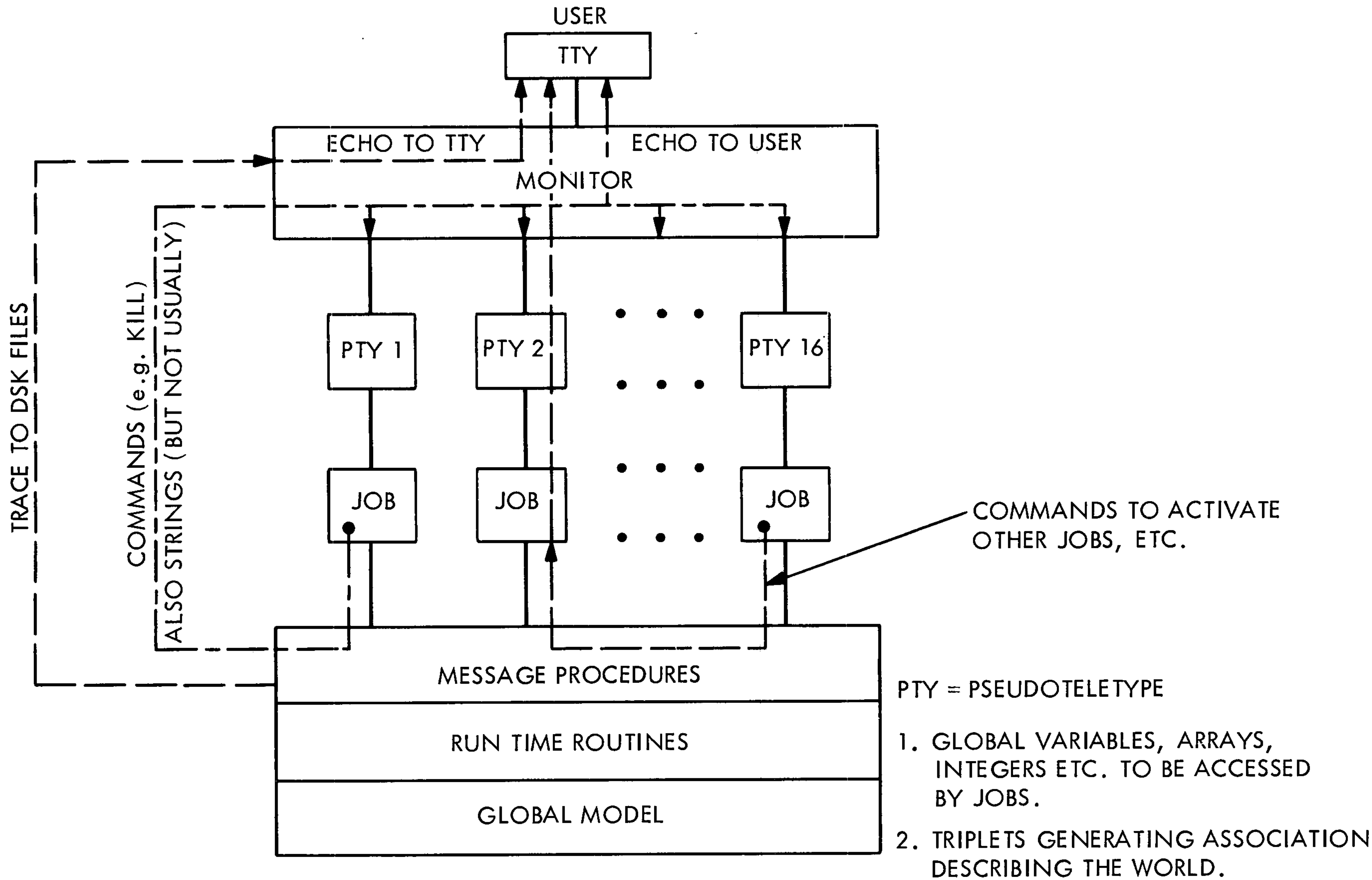

Fig. 2. Flow paths through Hand/Eye modules

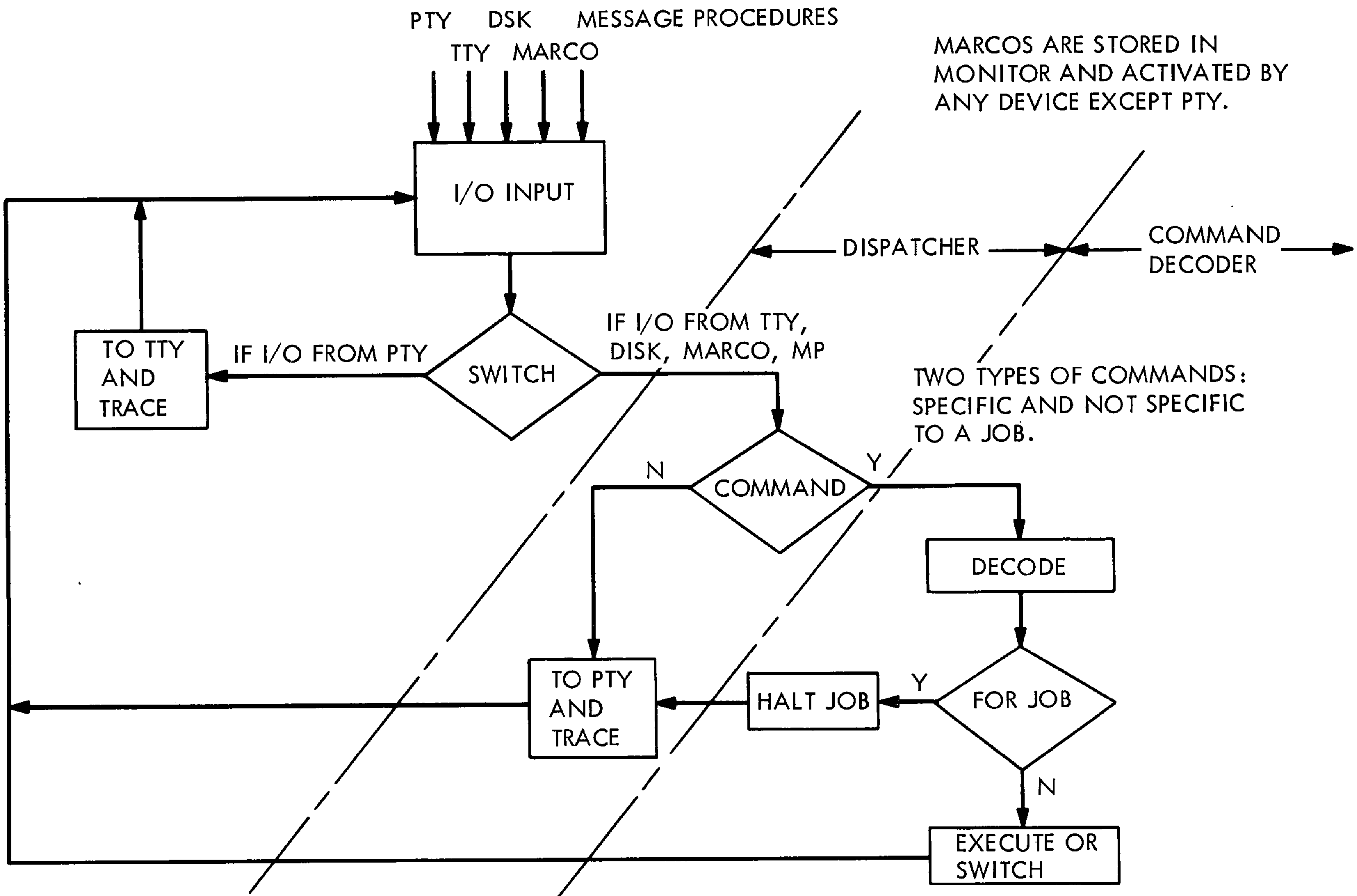

Fig. 3. HE monitor dispatcher, I/O, command decoder flow diagram

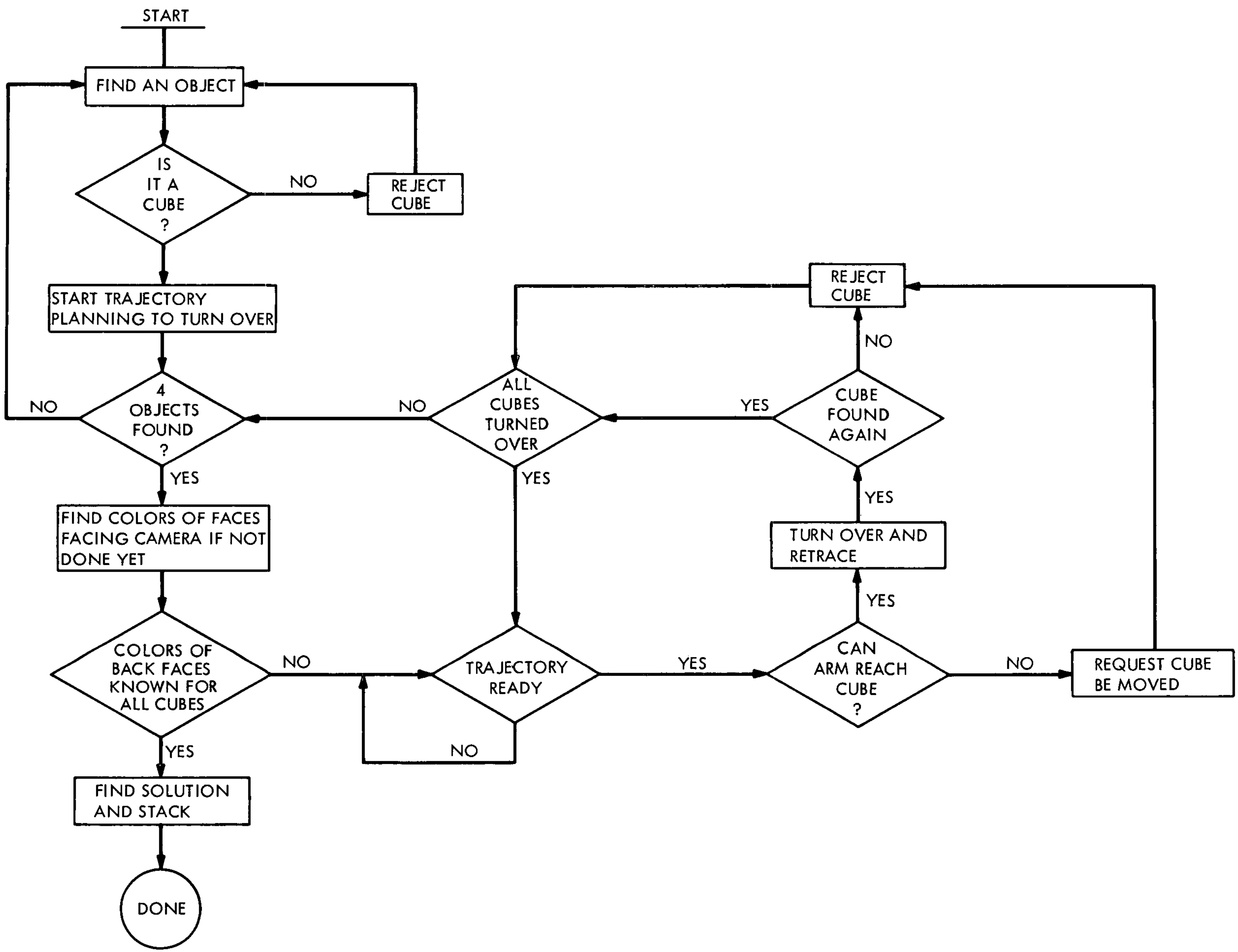

Fig. 4. Simplified flow diagram of program control (Instant Insanity puzzle)

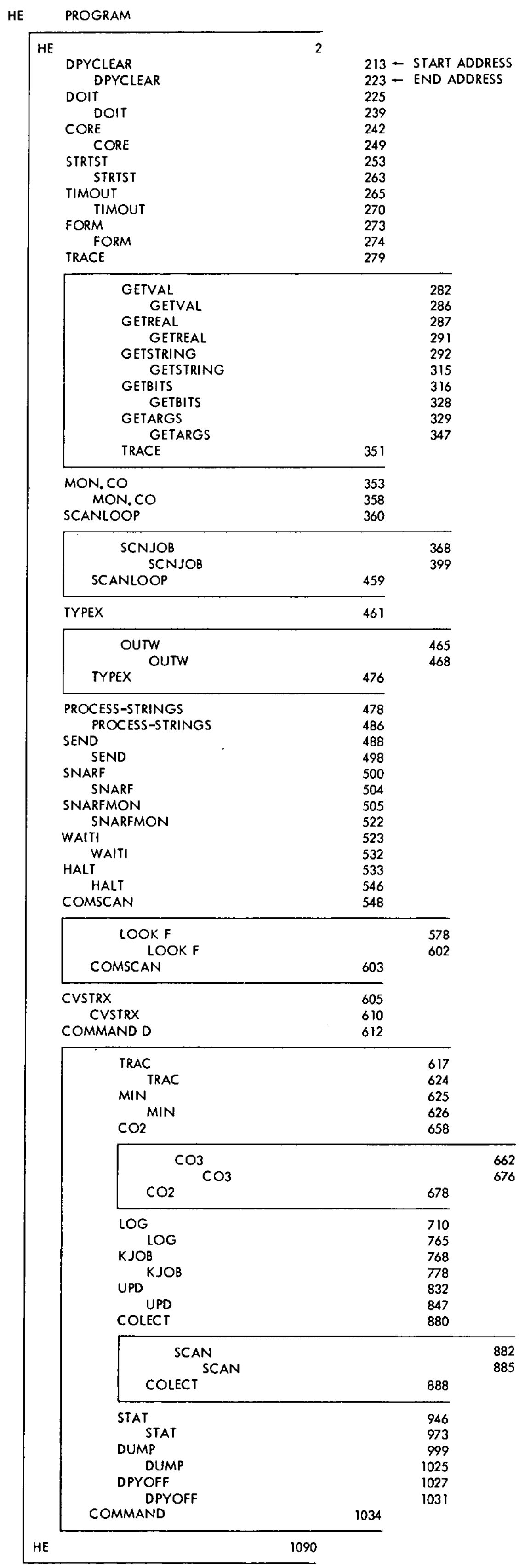

Fig. 5. Hand/Eye block structure

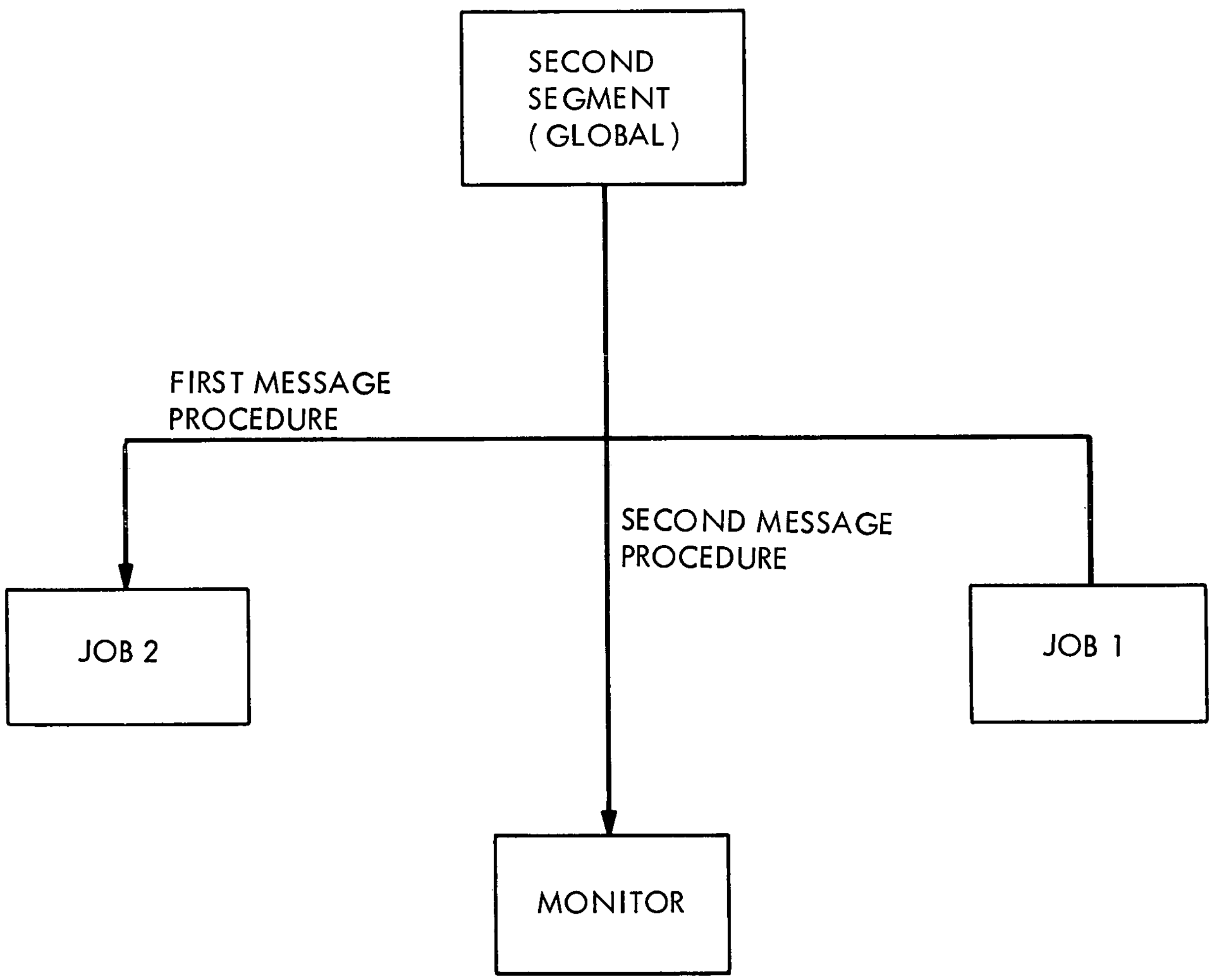

Fig. 6. Message procedure trace

APPENDIX A

HAND/EYE SYSTEM JOBS

The eight major jobs defined in the Hand/Eye system are the following:

<u>EDG</u>: Edge follower scans the TV's field of view, using a coarse raster, looking for edges. It then traces around the edges to find outline of object.

<u>SIM</u>: Simple body recognizer. It gets the corner coordinates of the objects in the global model and applies various tests to obtain a prediction as to what the object may be.

<u>CAM</u>: Changes the status of the TV camera, e.g., change lens, pan, tilt, pan and tilt, focus, focus and pan, focus and tilt, focus, pan and tilt, center.

<u>VER</u>: The verifier is called to determine whether or not an edge or line exists between TV coordinates (X1, Y1) and (X2, Y2). The value of the procedure is the confidence of the program in the existence of an edge.

<u>COL</u>: This procedure finds the colors of the visible face of each object.

<u>DRV</u>: Arm driver. The potentiometer readings generated by the arm solution program are obtained and the arm joints are servoed.

<u>GUN</u>: Driver for the region finder which prepares blobs for COMPLEX.

<u>CUR</u>: Curve fitter driver which tries to curve fit a set of blobs.

APPENDIX B

CALLI SYMBOLICS

DEC STANDARD

```
CX RESET,RESETUUO         ;0 RESET IO
CX DDTIN,DDTIN            ;1 EXT-GET DDT CHAR.
CX SETDDT,SETDDT          ;2 SETDDT LOC IN PROTECTED JOB DATA
CX DDTOUT,DDTOUT          ;3 EXT:SEND DDT CHAR.
CX DEVCHR,DVCHR           ;4 DEVICE CHARACTISTICS
CX DDTGT,CPOPJ            ;5 GET DDT MODE
CX GETCHR,DVCHR           ;6 DEVICE CHAR.(DIFF. NAME)
CX DDTRL,CPOPJ            ;7 RELEASE DDT MODE
CX WAIT,WAIT              ;10 WAIT TILL DEVICE INACTIVE
CX CORE,CORUUO            ;11 CORE UUO
CX EXIT,EXIT              ;12 EXIT
CX UTPCLR,UTPCLR          ;13 CLEAR DEC TAPE DIRECTORY
CX DATE,DATE              ;14 GET DATE
CX LOGIN,LOGIN            ;15 LOGIN
CX APRENB,APRENB          ;16 ENABLE APR FOR TRAPPING
CX LOGOUT,LOGOUT          ;17 LOGOUT
CX SWITCH,SWITCH          ;20 RETURN DATA SWITCHES
CX REASSIGN,REASSIGN      ;21 REASSIGN DEVICE TO ANOTHER JOB
CX TIMER,TIMER            ;22 RETURN JIFFY CLOCK TIME
CX MSTIME,MSTIME          ;23 RETURN TIME OF DAY IN MS
CX GETPPN,GETPPN          ;24 RETURN PROJECT-PROGRAMMER NUMBER
CX TRPSET,UUOERR          ;25 SET PI TRAP LOC, AND USER IO
CX TRPJEN,UUOERR          ;26 DISMISS INTERRUPT TO EXEC MODE
CX RUNTIM,JOBTIM          ;27 RETURN TOTAL JOB RUNNING TIME
CX PJOB,JOBNO             ;30 RETURN JOB NUMBER
CX SLEEP,SLEEP            ;31 SLEEP FOR N SECONDS, THEN RETURN TO USER
CX SETPOV,SETPOV          ;32 SET PUSH DOWN OVERFLOW TRAP
                          ;(FOR COMPATIBILITY ONLY)
CX PEEK,PEEK              ;33 PEEK INTO SYSTEM CORE.            ;JS
CX GETLIN,GETLN           ;34 GET NAME OF TTY
CX RUN,UUOERR             ;35 RUN COMMAND
CX SETUWP,setuwp          ;36 SET USER WRITE PROTECT
CX REMAP,remap            ;37 REDO CORE MAP
CX GETSEG,UUOERR          ;40 GET SEGMENT
CX GETTAB,UUOERR          ;41 GETTAB ILLEGAL AT STANFORD.
```

STANFORD DEFINED

```
CX SPCWAR,SPCWAR          ;0 READ SWITCH REGISTER ;JS
CX CTLV,CTLV              ;1 PUT TTY IN NON-DUPLEX MODE. ;JS
CX SETNAM,SETNAM          ;2 SET JOB NAME FOR SYSTAT
CX SPCWGO,SPCWGO          ;3 ANOTHER SPACEWAR UUO
```

```
CX SWAP,SYSRJB              ;4 RUN A JOB
CX EIOTM,EIOTM              ;5 ENTER IOT USER MODE
CX LIOTM,LIOTM              ;6 LEAVE IOT USER MODE
CX PNAME,PNAME              ;7 GET A DEVICE'S PHYSICAL NAME
CX UFBGET,UFBGET            ;10 GET A FAST BAND
CX UFBGIV,UFBGIV            ;11 RELEASE A FAST BAND
CX UFBCLR,FBFLUSH           ;12 RELEASE ALL FAST BANDS
CX JBTSTS,USTAT             ;13 GET JOB STATUS WORD OF A JOB
CX TTYIOS,TTYIOS            ;14 GET A JOB'S TELETYPES STATUS WORD
CX core2,core2              ;15 Funny core UUO for high segments
CX attseg,attseg            ;16 Attach high segment
CX detseg,detseg            ;17 Detach high segment
CX setpro,setpro            ;20 Change protection of high segment
CX segnum,segnum            ;21 get number of high segment
CX segsiz,segsiz            ;22
CX linkup,linkup            ;23
CX dismis,dismis            ;24
CX intnb,intnb              ;25 enable interrupts
CX intorm,intorm            ;26
CX intacm,intacm            ;27
CX intns,intns              ;30
CX intilp,intilp            ;31
CX intirq,intirq            ;32
CX intgen,intgen            ;33 generate an interrupt
CX uwait,uwait              ;34
CX debreak,debreak          ;35
CX setnm2,setnm2            ;36 set name of upper, if any
CX segnam,segnam            ;37 get name of upper, if any
CX IWAIT,IWAIT              ;40
CX uskip,uskip              ;41 Skip if a UWAIT really has to wait.
CX buflen,buflen            ;42 Return buffer length for a device
CX namein,namein            ;43 See if this Job name exists
CX slevel,setlvl            ;44 Set or get service level.
CX ienbw,ienbw              ;45 Enable interrupts and immediately go into
CX runmsk,runmsk            ;46 Sets processor run mask       wait state
LIST
```

APPENDIX C

A TRACE OF HAND/EYE SYSTEM EXECUTION

```
29 Mar 1972   14:09          TRAC53.DBG[2,KKP]

TTY→MON DISKIN HEMACR[II,HE]
DISK→MON          DEFINE EDGRUN
DISK→MACR         EDG:LOG
DISK→MACR         EDG:RUN DSK EDGE[II,HE]
DISK→MACR         EDG;GATER
DISK→MACR         DRV;
DISK→MACR
MON→TTY EDGRUN DEFINED
DISK→MON          DEFINE CURRUN
DISK→MACR         CUR:LOG
DISK→MACR         CUR:RUN DSK CURVE[II,HE]
DISK→MACR         CUR;GATER
DISK→MACR         DRV;
DISK→MACR
MON→TTY CURRUN DEFINED
DISK→MON          DEFINE CAMRUN
DISK→MACR         CAM:LOG
DISK→MACR         CAM:RUN DSK CAMERA[II,HE]
DISK→MACR         CAM;GATER
DISK→MACR         DRV;
DISK→MACR
MON→TTY CAMRUN DEFINED
DISK→MON          DEFINE IIRUN
DISK→MACR         DRV:LOG
DISK→MACR         DRV:RUN DSK IIDRV[II,HE]
DISK→MACR         DRV;GATER
DISK→MACR
MON→TTY IIRUN DEFINED
DISK→MON          DEFINE SIMRUN
DISK→MACR         SIM:LOG
DISK→MACR         SIM:RUN DSK SIMPLE[II,HE]
DISK→MACR         SIM;GATER
DISK→MACR         DRV;
DISK→MACR
MON→TTY SIMRUN DEFINED
DISK→MON          DEFINE COLRUN
DISK→MACR         COL:LOG
DISK→MACR         COL:RUN DSK COLOR[II,HE]
DISK→MACR         COL;GATER
DISK→MACR         DRV;
DISK→MACR
MON→TTY COLRUN DEFINED
DISK→MON          DEFINE VERRUN
DISK→MACR         VER:LOG
DISK→MACR         VER:RUN DSK VERIFY[II,HE]
DISK→MACR         VER;GATER
DISK→MACR         DRV;
DISK→MACR
MON→TTY VERRUN DEFINED
DISK→MON          DEFINE HANDRUN
DISK→MACR         HAND:LOG
DISK→MACR         :RUN DSK HAND[II,HE]
```

```
DISK→MACR       HAND;GATER
DISK→MACR       DRV;
DISK→MACR
MON→TTY HANDRUN DEFINED
DISK→MON        DEFINE MOVERUN
DISK→MACR       MOVE;LOG
DISK→MACR       ;RUN DSK MOVE[II,HE]
DISK→MACR       MOVE;GATER
DISK→MACR       DRV;
DISK→MACR
MON→TTY MOVERUN DEFINED
DISK→MON        DEFINE SETUP
DISK→MACR       ;IIRUN
DISK→MACR       ;;TRACE
DISK→MACR       ;;SET TYPE
DISK→MACR
MON→TTY SETUP DEFINED
DISK→MON        DEFINE ANDY
DISK→MACR       DRV;LOG
DISK→MACR       ;RUN DRIVER[H,JAM]
DISK→MACR       ;;SET TYPE
DISK→MACR       ;;TRACE
DISK→MACR       DRV;GATER
DISK→MACR
MON→TTY ANDY DEFINED
MON→TTY END DISKIN
TTY→MON SETUP
MACR→MON        IIRUN
MACR→MON        LOG
MACR→DRV        L
MACR→DRV          2/KKP
MON→TTY DRV LOGGED IN AS JOB 26
DRV→TTY
MACR→MON        RUN DSK IIDRV[II,HE]
MACR→DRV        RUN DSK IIDRV[II,HE]
DRV→TTY
MACR→DRV        GATER
MON→TTY END MACRO
DRV→TTY ,↑C
MACR→MON        TRACE
DRV→TTY
MACR→MON        SET TYPE
MON→TTY END MACRO
DRV→TTY ,SEGMENT LOGICAL NAME?
DRV→TTY
DRV→TTY
DRV→TTY
DRV→TTY
DRV→TTY
DRV→TTY
DRV→TTY
DRV→TTY
```

```
DRV→TTY
DRV→TTY
DRV→TTY
DRV→TTY UTILITY ROUTINES INITIALIZED
DRV→TTY *
TTY→MON CAMRUN
MACR→MON         LOG
MACR→CAM         L
MACR→CAM           2/KKP
MON→TTY CAM LOGGED IN AS JOB 27
CAM→TTY
MACR→MON         RUN DSK CAMERA[II,HE]
MACR→CAM         RUN DSK CAMERA[II,HE]
CAM→TTY
MACR→CAM         GATER
MACR→DRV
MON→TTY END MACRO
CAM→TTY ,↑C
CAM→TTY
TTY→MON EDGRUN
MACR→MON         LOG
MACR→EDG         L
MACR→EDG           2/KKP
MON→TTY EDG LOGGED IN AS JOB 28
CAM→TTY .SEGMENT LOGICAL NAME?
EDG→TTY
MACR→MON         RUN DSK EDGE[II,HE]
MACR→EDG         RUN DSK EDGE[II,HE]
CAM→TTY DATXFR: RETRIEVING DATA[1,SHY]1
EDG→TTY
MACR→EDG         GATER
CAM→TTY DATXFR: RETRIEVING DATA[1,SHY]2
MACR→DRV
MON→TTY END MACRO
TTY→MON CURRUN
MACR→MON         LOG
MACR→CUR         L
MACR→CUR           2/KKP
MON→TTY CUR LOGGED IN AS JOB 29
CAM→TTY DATXFR: RETRIEVING DATA[1,SHY]3
EDG→TTY .SEGMENT LOGICAL NAME?
CUR→TTY
MACR→MON         RUN DSK CURVE[II,HE]
MACR→CUR         RUN DSK CURVE[II,HE]
CAM→TTY DATXFR: RETRIEVING DATA[1,SHY]4
EDG→TTY *
CUR→TTY
MACR→CUR         GATER
CAM→TTY CAM_UPD: POTS TOO NOISY (13  2  13)
MACR→DRV
MON→TTY END MACRO
CUR→TTY ,↑C
```

```
CUR→TTY
TTY→MON SIMRUN
MACR→MON        LOG
MACR→SIM        L
MACR→SIM          2/KKP
MON→TTY SIM LOGGED IN AS JOB 30
CAM→TTY ...TYPE Y TO TRY AGAIN!
CUR→TTY ,SEGMENT LOGICAL NAME?
SIM→TTY
MACR→MON        RUN DSK SIMPLE[II,HE]
MACR→SIM        RUN DSK SIMPLE[II,HE]
SIM→TTY
MACR→SIM        GATER
MACR→DRV
MON→TTY END MACRO
SIM→TTY ,↑C
SIM→TTY
TTY→MON COLRUN
MACR→MON        LOG
MACR→COL        L
MACR→COL          2/KKP
MON→TTY COL LOGGED IN AS JOB 31
SIM→TTY ,SEGMENT LOGICAL NAME?
COL→TTY
MACR→MON        RUN DSK COLOR[II,HE]
MACR→COL        RUN DSK COLOR[II,HE]
SIM→TTY WARNING: TWO PROGRAMS WITH ITEMS IN THEM
COL→TTY
MACR→COL        GATER
MACR→DRV
MON→TTY END MACRO
TTY→CAM
CAM→TTY CAM←ACTIVATED
COL→TTY ,SEGMENT LOGICAL NAME?
TTY→MON STAT
DRV→TTY 26      II        IIDRV      2,KKP    IOWQ    28K   010,383     010,38
CAM→TTY 27      CAM       CAMERA     2,KKP    IOWQ    14K   010,683     010,68
EDG→TTY 28      EDGE      EDGE       2,KKP    INTWQ   35K   010,316     010,31
CUR→TTY 29      CURVE     CURVE      2,KKP    IOWQ    18K   010,200     010,20
SIM→TTY 30      SIMP      SIMPLE     2,KKP    IOWQ    33K   010,333     010,33
COL→TTY 31      COL       COLOR      2,KKP    IOWQ    29K   010,383     010,38
MON→TTY
TOTAL CORE = 191K   UPPER SEG=18K   MAX=65K  12K LEFT
TTY→MON TRACE
TTY→MON SET TYPE
TTY→EDG DEBUG EDGE ON
EDG→TTY *
TTY→DRV BLOB←GETEDGE(0)
DRV→TTY SENDING INSIDE NIL
49954733        MESSAGE TRACE: II    EDGE   INSIDE IvV
DRV→TTY WAITING FOR RESPONSE INSIDE
EDG→TTY DAC SET AT       62    AD=    2711
```

```
EDG→TTY DAC SET AT        1   AD=    1884
TTY→MON STAT
EDG→TTY 28      EDGE      EDGE       2,KKP   RUNQ    35K   Ø:4,616    Ø:4,30
EDG→TTY DAC SET AT       31   AD=    1892
EDG→TTY DAC SET AT       46   AD=    1897
EDG→TTY DAC SET AT       54   AD=    2181
EDG→TTY DAC SET AT       5Ø   AD=    1907
EDG→TTY DAC SET AT       52   AD=    2038
EDG→TTY DAC SET AT       51   AD=    1974
EDG→TTY AUTO TARGET SET AT        5Ø
EDG→TTY REINIT TCLIP=       3  BCLIP=      4
EDG→TTY DAC SET AT       5Ø   AD=    1903
EDG→TTY CLIPSET TCLIP=      7  BCLIP=       7
EDG→TTY XTENT OK
EDG→TTY KKP:FOUND MATCHING END
5Ø15Ø6ØØ        MESSAGE TRACE: EDGE   II  RESPONSE "FIND" 3788 Ø
DRV→TTY WAITING FOR RESPONSE INSIDE
5Ø158716        MESSAGE TRACE: EDGE   II  RESPONSE "INSIDE" 4Ø28 -2
DRV→TTY *
TTY→DRV BLOB=
DRV→TTY BLOB NOT RECOGNIZED OR ILLEGAL
DRV→TTY
DRV→TTY *
TTY→DRV BLOB
TTY→DRV
DRV→TTY = (BLOB_1)
DRV→TTY *
TTY→DRV BLOB←INNER(BLOB)
DRV→TTY SENDING FINE BLOB_1
5Ø2Ø7116        MESSAGE TRACE: II   EDGE  FINE IvV
DRV→TTY WAITING FOR RESPONSE FINE
5Ø21Ø266        MESSAGE TRACE: EDGE   CURVE  CURVE_FIT FAR
EDG→TTY
EDG→TTY ØØØØØ6 WORDS COLLECTED - GARCOL
EDG→TTY
EDG→TTY ØØØØØØ WORDS COLLECTED - GARCOL
EDG→TTY KKP:POINT SEEN BEFORE
EDG→TTY DELETED
EDG→TTY CLIPSET TCLIP=                 7  BCLIP=                7
EDG→TTY CLIPSET TCLIP=                 7  BCLIP=                7
EDG→TTY CLIPSET TCLIP=                 7  BCLIP=                7
EDG→TTY KKP:LOOPING
EDG→TTY KKP: SCAN REVERSED
EDG→TTY CLIPSET TCLIP=                 7  BCLIP=                7
EDG→TTY KKP: ACCOM FAILED
EDG→TTY KKP: OBJECT SEEN
EDG→TTY KKP:HIT CURRENT OBJECT
EDG→TTY DAC SET AT                    53   AD=            2124
EDG→TTY DAC SET AT                    5Ø   AD=            1915
EDG→TTY CLIPSET TCLIP=                 7  BCLIP=                7
EDG→TTY DAC SET AT                    53   AD=            2125
EDG→TTY DAC SET AT                    56   AD=            2331
```

```
EDG←TTY CLIPSET TCLIP=                     4  BCLIP=                  6
EDG←TTY KKP; OBJECT SEEN
EDG←TTY KKP;HIT CURRENT OBJECT
EDG←TTY KKP; SCAN REVERSED
EDG←TTY KKP;HIT CURRENT OBJECT
EDG←TTY DAC SET AT                    53   AD=                  2130
EDG←TTY DAC SET AT                    50   AD=                  1928
EDG←TTY CLIPSET TCLIP=                     7  BCLIP=                  7
EDG←TTY DAC SET AT                    53   AD=                  2128
EDG←TTY DAC SET AT                    56   AD=                  2320
EDG←TTY DAC SET AT                    53   AD=                  2130
EDG←TTY DAC SET AT                    50   AD=                  1922
EDG←TTY CLIPSET TCLIP=                     7  BCLIP=                  7
EDG←TTY CLIPSET TCLIP=                     7  BCLIP=                  7
EDG←TTY CLIPSET TCLIP=                     7  BCLIP=                  7
EDG←TTY KKP; ACCOM FAILED
EDG←TTY KKP; SCAN REVERSED
EDG←TTY KKP;LOOPING
EDG←TTY DAC SET AT                    53   AD=                  2130
EDG←TTY DAC SET AT                    56   AD=                  2327
EDG←TTY KKP; HIT END OF PREVIOUS OBJECT
EDG←TTY KKP; SCAN REVERSED
EDG←TTY KKP;HIT CURRENT OBJECT
EDG←TTY KKP; TRY FOR MORE
EDG←TTY KKP;HIT CURRENT OBJECT
50416033        MESSAGE TRACE; EDGE   CURVE  CURVE_FIT FAR
EDG←TTY DAC SET AT                    53   AD=                  2132
EDG←TTY CLIPSET TCLIP=                     2  BCLIP=                  4
EDG←TTY KKP; ACCOM FAILED
EDG←TTY KKP; OBJECT SEEN
EDG←TTY KKP;HIT CURRENT OBJECT
EDG←TTY KKP; SCAN REVERSED
EDG←TTY KKP;HIT CURRENT OBJECT
EDG←TTY KKP;POINT SEEN BEFORE
EDG←TTY DELETED
EDG←TTY DAC SET AT                    56   AD=                  2327
EDG←TTY DAC SET AT                    53   AD=                  2132
50466650        MESSAGE TRACE; EDGE   CURVE  CURVE_FIT FAR
EDG←TTY KKP;POINT SEEN BEFORE
EDG←TTY DELETED
EDG←TTY DAC SET AT                    56   AD=                  2336
EDG←TTY DAC SET AT                    53   AD=                  2131
50492933        MESSAGE TRACE; EDGE   CURVE  CURVE_FIT FAR
EDG←TTY DATA MISSED - TV
EDG←TTY HUNG DEVICE AD
EDG←TTY TYPE C<CR> TO CONTINUE, ANYTHING ELSE <CR> TO RETRY
TTY←EDG
EDG←TTY HUNG DEVICE AD
EDG←TTY TYPE C<CR> TO CONTINUE, ANYTHING ELSE <CR> TO RETRY
EDG←TTY HUNG DEVICE AD
EDG←TTY TYPE C<CR> TO CONTINUE, ANYTHING ELSE <CR> TO RETRY
TTY←EDG
```

```
EDG→TTY HUNG DEVICE AD
EDG→TTY TYPE C<CR> TO CONTINUE, ANYTHING ELSE <CR> TO RETRY
EDG→TTY HUNG DEVICE AD
EDG→TTY TYPE C<CR> TO CONTINUE, ANYTHING ELSE <CR> TO RETRY
TTY→EDG C
EDG→TTY DAC SET AT                        50   AD=               -1
TTY→MON S
TTY→EDG S
EDG→TTY
50586750        MESSAGE TRACE: EDGE   II  RESPONSE "FINE" 4028 -1
EDG→TTY *
DRV→TTY *
TTY→EDG BLOB
EDG→TTY COM ERR BLOB
EDG→TTY *
TTY→DRV BLOB
DRV→TTY = ()
DRV→TTY *
TTY→EDG REJECT -1
EDG→TTY REJECT 4028 -1
EDG→TTY *
TTY→EDG BLOB←GETEDGE(1)
EDG→TTY COM ERR BLOB←GETEDGE(1)
EDG→TTY *
TTY→DRV BLOB←GETEDGE(1)
DRV→TTY SENDING FIND NIL
50639533        MESSAGE TRACE: II   EDGE  FIND IvV
DRV→TTY WAITING FOR RESPONSE FIND
EDG→TTY COLOR WHEEL IS HUNG! RETRY OR CONTINUE (R OR C)
TTY→EDG R
EDG→TTY DAC SET AT         1   AD=    1883
EDG→TTY DAC SET AT        48   AD=    1886
EDG→TTY DAC SET AT        49   AD=    1897
EDG→TTY DAC SET AT        50   AD=    1918
EDG→TTY AUTO TARGET SET AT         50
EDG→TTY REINIT TCLIP=        3  BCLIP=        4
EDG→TTY PARITY ERROR, IN YOUR CORE IMAGE!
EDG→TTY LOC= 7020
EDG→TTY ↑C
EDG→TTY
EDG→TTY .?
EDG→TTY ERROR IN JOB 28
EDG→TTY ILL MEM REF AT USER 7020
EDG→TTY ↑C
EDG→TTY
TTY→MON S
TTY→EDG S
EDG→TTY .↑C
EDG→TTY
EDG→TTY .
EDG→TTY DAC SET AT         1   AD=    1851
EDG→TTY DAC SET AT        25   AD=    1872
```

```
EDG→TTY DAC SET AT        37   AD=    1883
EDG→TTY DAC SET AT        50   AD=    1918
EDG→TTY AUTO TARGET SET AT       50
EDG→TTY REINIT TCLIP=       3  BCLIP=       4
EDG→TTY ?
EDG→TTY ERROR IN JOB 28
EDG→TTY ILL MEM REF AT USER 7020
EDG→TTY ↑C
EDG→TTY
TTY→MON RUN EDGE[II,HE]
TTY→EDG RUN EDGE[II,HE]
EDG→TTY ,↑C
EDG→TTY
EDG→TTY ,SEGMENT LOGICAL NAME?
TTY→EDG GATER
EDG→TTY DAC SET AT         1   AD=    1847
EDG→TTY DAC SET AT        25   AD=    1878
EDG→TTY DAC SET AT        37   AD=    1885
EDG→TTY DAC SET AT        50   AD=    1918
EDG→TTY AUTO TARGET SET AT       50
EDG→TTY REINIT TCLIP=       3  BCLIP=       4
EDG→TTY XTENT OK
EDG→TTY
EDG→TTY 000000 WORDS COLLECTED - GARCOL
EDG→TTY KKP;FOUND MATCHING END
50821600         MESSAGE TRACE: EDGE   II  RESPONSE "FIND" 3785 0
50822183         MESSAGE TRACE: EDGE   II  RESPONSE "FIND" 4028 -1
DRV→TTY WAITING FOR RESPONSE FIND
EDG→TTY *
DRV→TTY *
TTY→DRV BLOB←CURVE(BLOB)
DRV→TTY SENDING FIT BLOB_2
50832150         MESSAGE TRACE: II   EDGE  FIT IvV
DRV→TTY WAITING FOR RESPONSE FIT
50834033         MESSAGE TRACE: EDGE   CURVE  CURVE_FIT FAR
50841783         MESSAGE TRACE: EDGE   II  RESPONSE "FIT" 3785 0
DRV→TTY *
TTY→DRV REJ←
DRV→TTY *
TTY→DRV OBJ←SIMPLE(BLOB,(ALL),REJ)
DRV→TTY SENDING SIMP_FIT BLOB_2
50862433         MESSAGE TRACE: II   SIMP  SIMP_FIT ItV 0 FLDIVR
SIM→TTY I AM NOW IN SIMPLE
SIM→TTY NUMBER OF CORNERS IS          6
SIM→TTY ITS'S A RECTANGULAR PARALLELEPIPED.
SIM→TTY INSTANCE TRANSFORM FROM SIMPLE
SIM→TTY -.957483       -.288490       .000000       27.2563
SIM→TTY  .288490       -.957483       .000000       27.6449
SIM→TTY  .000000        .000000       1.00000       .625000
SIM→TTY  .000000        .000000       .000000       1.00000
DRV→TTY *
TTY→DRV DISP_OBJ(OBJ,1)
```

```
DRV→TTY = *** NO VALUE ***
DRV→TTY *
TTY→MON RESET DI
TTY→DRV COLFIND(OBJ)
DRV→TTY COLFIND NOT RECOGNIZED OR ILLEGAL
DRV→TTY OBJ)
DRV→TTY *
TTY→DRV COL_FIND(OBJ)
DRV→TTY COL_FIND NOT RECOGNIZED OR ILLEGAL
DRV→TTY OBJ)
DRV→TTY *
TTY→MON UPDATE
```


NASA — JPL — Coml., L.A., Calif.