

ERROR RECOVERY IN ROBOT SYSTEMS

Thesis by
Sankaran Srinivas

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1977

(Submitted December 15, 1976)

DEDICATION

To Lorraine, Vani, Sriram, Amma, Appa,
and to all the others in my family.

ACKNOWLEDGEMENTS

I would like to thank:

Prof. G.D.McCann for his help and support during my years at Caltech,

Dr. M.Weinstein who got me started in this area of research, pointed out the importance of the problem of error recovery, suggested many ideas, and provided helpful guidance and encouragement,

Dr. W.M.Whitney for helping me formulate my ideas through many and Mr. L.Friedman discussions and for providing the right encouragement at needed moments,

Dr. G.Ingargiola, for their important role in teaching me many Prof. F.B.Thompson important concepts of computer and information and Prof. P.B.Hansen science,

Prof. H.J.Ryser for taking an active interest in certain combinatorial problems arising in connection with my work and for providing many ideas on how it should be tackled,

Prof. R.H.Cannon for making some useful suggestions regarding the presentation of the general contributions of this thesis,

Shriram Udupa my fellow students, for many interesting, and Scott Roth speculative, and valuable discussions, and for making my stay at Caltech both memorable and enjoyable,

Lorraine Alvarez without whose help, support, encouragement, and confidence in my abilities, this thesis would never have been completed,

Caltech and JPL for providing me with financial assistance in the form of tuition scholarships, graduate teaching assistantships, and graduate research assistantships.

ABSTRACT

This dissertation addresses itself to the problem faced by a robot in recovering from failures during execution of a task. Failures occur partly because sensory information is inaccurate, partly because effectors do not always perform as expected, and partly because the domain in which the robot operates cannot be characterized exactly. Robot systems with automated planners have traditionally dealt with the problem of error recovery by merely replanning to achieve the desired goal, without attempting to characterize the failure in any way whatsoever.

The central idea in this thesis is that planning recovery from failures has its own special techniques, distinct from those used in conventional planning systems. Two viewpoints, looking at the past for an explanation of the failure, and looking at the current situation to attempt a characterization of the failure state, provide powerful heuristics for error recovery. This thesis suggests that these heuristics can be formalized as failure reason analysis and multiple outcome analysis, and that knowledge relevant for such analysis can be provided through a failure reason model and a multiple outcome model associated with each action.

The failure reason model provides a means for representing knowledge about why actions fail, like bumping into an object to be grasped because of servoing errors or because of inaccurate information about the location of the object. The model also provides knowledge

required for distinguishing between the different reasons for failure. Finally, it includes recommendations of corrective actions to be taken if failure is attributed to a specific reason. This model is used in failure reason analysis in building a failure tree representing possible explanations of the failure. The explanations represented in the tree are then used in planning recovery.

The multiple outcome model provides a way of representing the possible outcomes of an action, like bumping onto the object or bumping onto the ground in the immediate vicinity of the object, ignoring the fact that these outcomes could be the result of several different reasons. Knowledge required to distinguish between different outcomes is provided as part of the model. In cases where the immediately available information is inadequate to identify the outcome of an action, the multiple outcome model provides a basis for executing actions to serve as information gathering steps. The novel feature here is that information gathering is directed by specific expectations about the state of the world.

A computer implementation of a program called MEND has provided a medium for exploring the above idea. MEND has been designed to automate recovery from failures in simple manipulation tasks to be performed by the JPL robot, but the techniques used in MEND have greater generality. A first implementation of MEND established the basis of this investigation. A second version, which has been designed to correct some limitations of the first version, has not yet been fully implemented and integrated with the JPL robot system.

The techniques of planning recovery from failures through failure reason analysis and multiple outcome analysis are contributions to the subject of robotics. More importantly, however, the problem of error recovery is recognized to be a member of a larger class of problems involving knowledge representation and common sense reasoning, both of which are core topics in the study of artificial intelligence. The solution presented in this thesis makes some new contributions to these core topics.

TABLE OF CONTENTS

1 INTRODUCTION	1
1.1 Problem Description	1
1.2 An overview of MEND	2
1.3 Other approaches to the problem	9
1.4 An overview of the report	14
2 THE JPL ROBOT	15
2.1 Introduction	15
2.2 JPL program objectives	15
2.3 JPL robot hardware	16
2.4 Tasks and goals	20
2.5 System components and structural organization	22
2.6 Example of execution	27
2.7 Errors and a module for error recovery	28
3 A DESCRIPTION OF MEND	30
3.1 Introduction	30
3.2 The internal structure of MEND	31
3.3 World model	36
3.4 Knowledge about actions	41
3.5 The execution trace	47

4 FAILURE REASON ANALYSIS	50
4.1 Introduction	50
4.2 A classification of failure reasons	50
4.3 The failure tree	53
4.4 Building a failure tree	55
4.5 The failure reason model	58
4.6 Scenarios	66
4.7 Comments and comparisons	77
5 MULTIPLE OUTCOME ANALYSIS	81
5.1 Introduction	81
5.2 An example : MOVE-HAND-TO-GRASP	82
5.3 A second example : GRASP	92
5.4 Failure reason analysis versus multiple outcome analysis	94
5.5 Integration of the two schemes	95
6 SUMMARY AND CONCLUSIONS	97
6.1 Introduction	97
6.2 Historical note	97
6.3 A discussion of the approach	98
6.4 Features of the solution	100
6.5 Contributions	103
6.6 Future work	104

7 REFERENCES	107
8 APPENDIX-1	112
9 APPENDIX-2	117

INTRODUCTION

1.1 PROBLEM DESCRIPTION

A robot performs a task by executing a plan of actions. During execution the robot may encounter a variety of unforeseen circumstances, some of which prevent successful completion of the task. Even simple actions such as reaching out to grasp a rock cannot be guaranteed to succeed in the absence of dynamic sensory feedback as in visual monitoring of manipulation tasks. The hand may bump onto the rock because of manipulation errors or because of erroneous information about the location of the rock. Failures such as these occur because sensors and effectors are inaccurate, and because the world in which the robot operates cannot be characterized exactly.

Our goal is to design a robot in such a way that it can recover from failures in a graceful and intelligent manner. Robot systems with automated planners have traditionally dealt with the problem of error recovery by merely replanning to achieve the desired goal, without attempting to characterize the failure in any way whatsoever.

The central idea in this thesis is that planning recovery from failures has its own special techniques, distinct from those used in conventional planning systems. The techniques of recovery planning are based on knowledge about failures -- why they occur, what can be done about them, etc. Two viewpoints, looking at the past for an explanation of the failure, and looking at the current situation to attempt a

characterization of the failure state, provide powerful heuristics for error recovery.

A computer implementation of a program called MEND has provided a medium for exploring the above idea. MEND has been designed to automate recovery from failures in simple manipulation tasks to be performed by the JPL robot, but the techniques used in MEND have greater generality. A first version of MEND established the basis of the investigation reported here. A second version, which has been designed to correct some limitations of the first version, has not yet been fully implemented and integrated with the JPL robot system.

1.2 AN OVERVIEW OF MEND

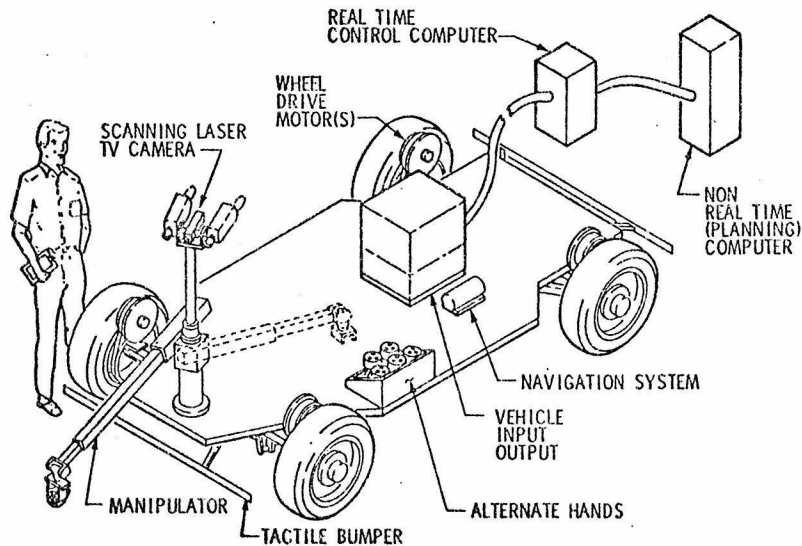


FIGURE 1.1

Let us consider an example to illustrate the problems involved in recovery from failures, and some of the features of the solution incorporated in MEND. The robot hardware that is controlled by MEND consists of two TV cameras, a laser rangefinder, a manipulator, and a vehicle [Figure 1.1]. Suppose that the robot is given the simple task of picking up a rock within immediate reach. Performing this task involves locating the rock using the TV cameras (possibly using the laser rangefinder to augment the camera data), positioning the hand around the rock, and grasping the rock [Figure 1.2]. Assume that the hand bumps into the rock when attempting to position the hand around the rock.

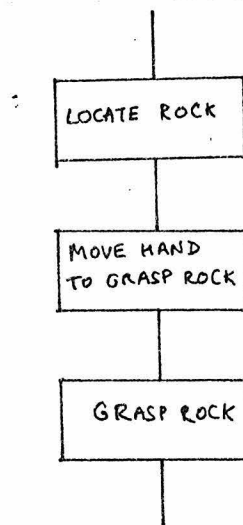


FIGURE 1.2

Any robot system designed to recover from failures must be able to detect them. In the above example, the manipulator system triggers the failure, bringing MEND into action. In other cases, MEND has to make explicit checks of certain conditions to detect whether or not something has gone wrong. MEND's execution monitoring is pragmatic in the sense

that only those conditions that are easy to check on the basis of the available information are tested. A consequence of this lack of comprehensiveness is that failures can propagate down the plan and may be detected only at a subsequent step. This places an additional burden on the recovery scheme. At this point we merely note the intimate relationship between execution monitoring and error recovery.

Having detected a failure, MEND is confronted with the problem of dealing with the unexpected event. Two general heuristics provide guidelines to planning recovery.

1.2.1 Failure reason analysis

The first heuristic suggests that recovery actions can be found by determining why the failure occurred. The process of finding an explanation for the failure is termed failure reason analysis. Knowledge necessary for failure reason analysis is provided through a failure reason model associated with each action. The failure reason model represents knowledge about the different reasons for failure of an action, the way in which they may be distinguished, and finally about what can be done to recover from a specific kind of failure.

MEND represents possible explanations for a failure in a structure called the failure tree, an example of which is shown in Figure 1.3. The tree consists of a linked set of failure nodes and action nodes. There are four basic types of failure reasons that are dealt with in MEND and they are : operational errors, information errors, precondition errors, and constraint errors. Each failure node is

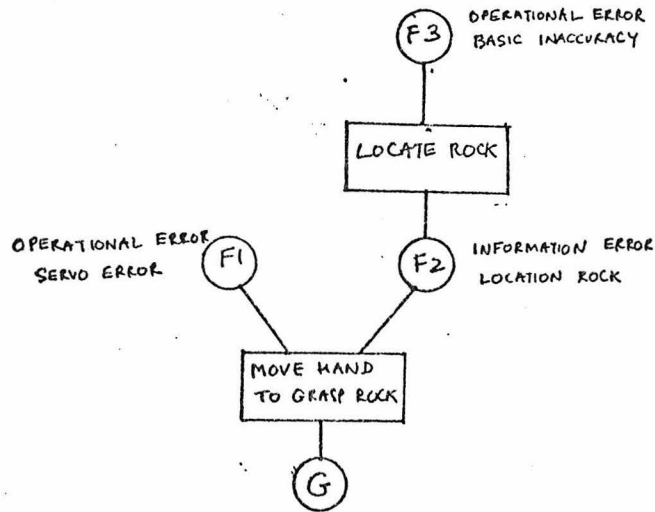


FIGURE 1.3

associated with a failure type and a specific reason for failure. Action nodes are linked to an action in the plan being executed. An example of an explanation represented in Figure 1.3 is that the goal WITHIN-GRASP (node G) was never achieved by action MOVE-HAND-TO-GRASP, because of an operational SERVO-ERROR (node F1).

Analysis of failures can now be thought of as the process of limiting the set of all possible explanations to the specific one which applies in a particular situation. MEND does this by limiting the set of failure reasons to be considered at each action node in the failure tree. Preconditions and constraints that have been independently verified can be eliminated from consideration as possible reasons for failure. More importantly, the distinctive features of the manifestation of the failure are used to limit the failure reason set. Figure 1.4 shows, for instance, a simple test and the implications of its results in eliminating some failure reasons from consideration.

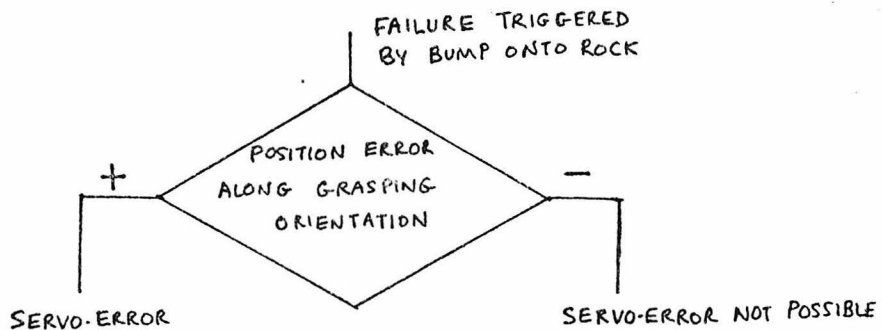


FIGURE 1.4

Once an explanation has been found, recommendations of corrective steps can lead MEND to successfully planning recovery from failures. For instance, if the failure is attributed to SERVO-ERROR [Figure 1.3], then recovery can be simply achieved by repositioning of the hand as shown in Figure 1.5.

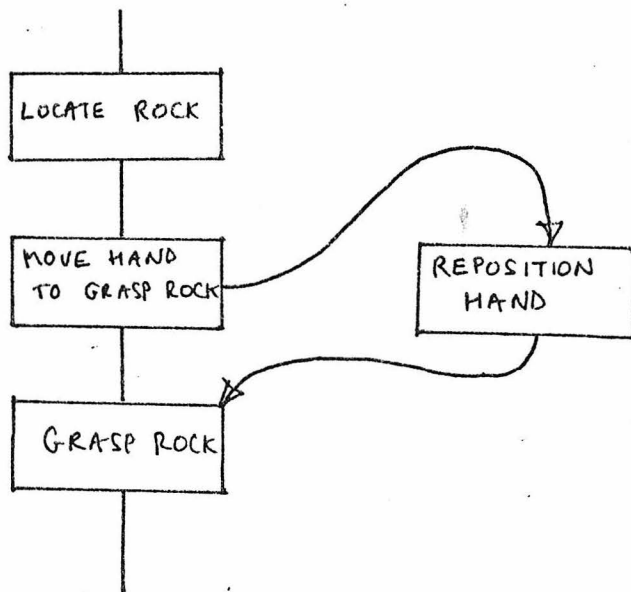


FIGURE 1.5

1.2.2 Multiple outcome analysis

The second heuristic ignores the reason for failure and suggests that recovery strategies can be found by determining what the outcome of an action is. This characterization is formalized as multiple outcome analysis and is directed by a multiple outcome model.

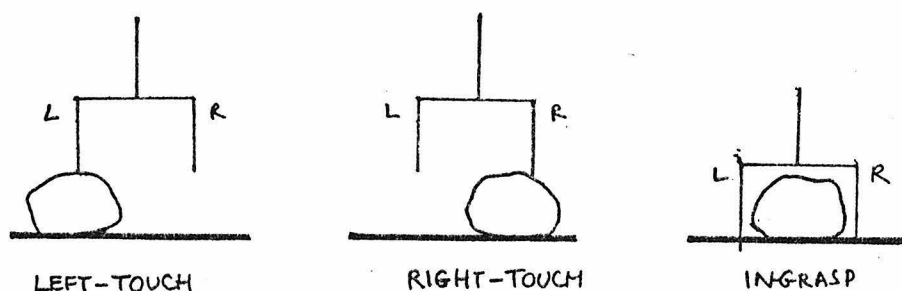
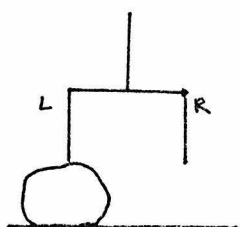


FIGURE 1.6

Continuing with our example, MEND finds from the multiple outcome model that some of the possible outcomes of positioning the hand around the rock are as shown in Figure 1.6. Either the fingers could have bumped onto the object, or the hand could have bumped onto the ground because of overshoot. The outcomes could be the result of servoing errors, object location errors, or a combination of these, but this is unimportant for multiple outcome analysis. MEND attempts to find the actual outcome of an action when failure has occurred by running a series of tests. These tests are based on the fact that each outcome is characterized by a set of conditions that must be true or false for that situation. To illustrate an example of such a set of conditions is shown in Figure 1.7. The important thing to note is that these

conditions can be checked on the basis of immediately available information.



LEFT TOUCH SENSOR ON
RIGHT TOUCH SENSOR CANNOT BE ACTIVATED
HAND ON OBJECT

FIGURE 1.7

The available information may not always suffice to identify the failure state. In such cases the multiple outcome model directs MEND in the execution of actions to collect the necessary information. Figure 1.8 shows a simple action and its functions in distinguishing between outcomes.

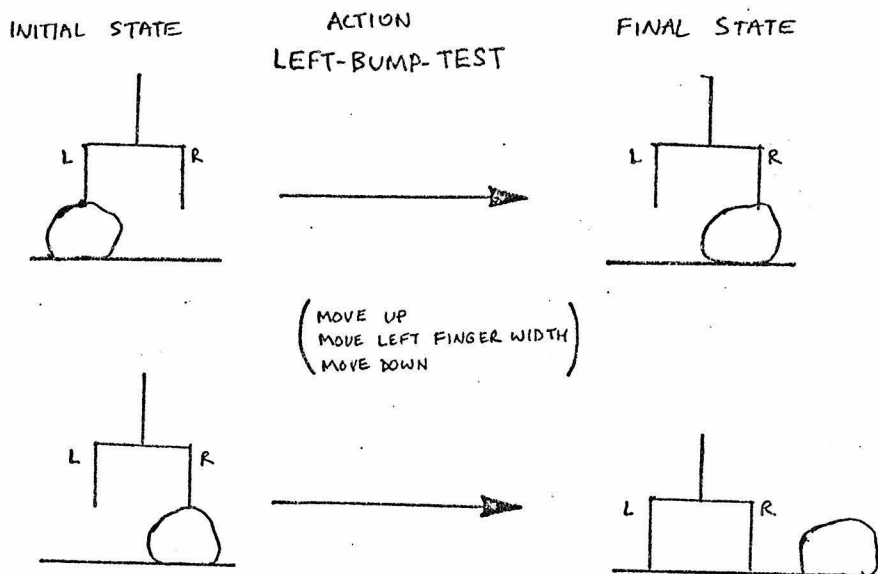


FIGURE 1.8

Finally, the multiple outcome model includes recommendations for recovery strategies for each possible outcome. An example of this is shown in Figure 1.9.

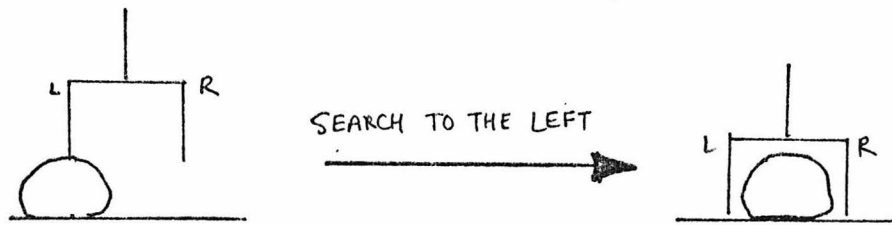


FIGURE 1.9

1.3 OTHER APPROACHES TO THE PROBLEM

The problem of getting robots to do useful things has been tackled in two distinct ways, and each of these approaches has implications for the way in which failures are handled. We shall refer to the two approaches as the higher level language approach and the planning approach. The distinction between the two methodologies becomes blurred in certain systems. Nevertheless it is a useful distinction for our discussion.

In the higher level language approach, each robot task requires the writing of a program. For instance, in the Stanford Hand-Eye System [Feldman 71a, 71b], every perceptual and manipulation task required the writing of a strategy control program. The strategy control program

could be designed to deal with failures. However, the system itself had no knowledge about failures and how to deal with them. Continuing studies at Stanford have led to the development of AL [Finkel 74], a specialized language for automated assembly tasks. Here again, failures can be dealt with by explicit programming for each task. Subroutines can encapsulate some recovery strategies, but incorporating knowledge about failures in subroutines has many limitations. For example, it becomes difficult (or even impossible) for the system to analyze failures and attribute the failure to a previous action that did not produce the desired result. In MEND such analysis becomes feasible because of a more explicit representation of knowledge about failures.

Consider now the planning approach. Here tasks are specified as a goal state to be achieved. An automated planner finds a sequence of steps to achieve the goal. In systems of this sort, failure during execution can be dealt with by replanning from the failure state to the goal state. It would seem that this provides a natural solution to the problem.

However, there is a basic problem in this whole approach that makes it difficult to deal with failures in a direct way. If we consider WITHIN-GRASP(ROCK) to be the desired goal state of positioning the hand to grasp an object, a failure such as bumping into the rock will be responded to by replanning to achieve the goal. It would not be difficult for the planning system to determine that this can be done by locating the rock, and then positioning the hand. This is fine, but such a system would be incapable of recognizing that simple

repositioning of the hand would be adequate if the failure were the result of servoing errors. To state it in more general terms, modelling actions merely in terms of preconditions and postconditions is not adequate in analyzing what caused the failure and what can be done about it. Such analysis is needed to construct practical recovery strategies.

A number of systems falling within the planning paradigm are described, in spite of the fact that the above criticism can be levelled against them. They are described because they embody some important concepts relevant to our discussion.

The planning approach is exemplified by the SHAKEY system [Raphael 71, Fikes 71]. Plans produced by STRIPS, the SHAKEY planner, were structured into triangle tables. The triangle tables gave precise meaning to the concept of a kernel -- the set of conditions that were relevant to the execution of the plan. Specifically, the important property is that the truth of the kernel associated with each action in a plan ensured that the subsequent steps would succeed (barring operational failures). Furthermore, this concept allowed SHAKEY to be clever in replanning from failures by attempting to achieve intermediate goal states.

Nilsson[73] has dealt briefly with the problem of error recovery in studying methods for integrating planning and execution. He identifies two kinds of events -- failures and surprises -- that a robot must deal with in a dynamic and uncertain world. His notion of failures includes execution time failures which we are concerned with, but also encompasses failures in planning. Execution time failures were handled

by the simple means of repeating the failed action, but the system had no good understanding of whether this would work or not.

The notion of surprises is a consequence of a dynamic world in which other agents play a role in changing the world. Nilsson suggests that these problems be tackled by the use of demons set up to watch for certain conditions. The demon would transfer control to a higher level executive when its activating conditions were made true. The hope was that the higher level executive would be able to deal with the surprise since it was the one which set up the demon.

Surprises are dealt with in an interesting manner by Hayes[75]. His system uses a representation of robot plans that make explicit the relationship between decisions and subgoals. New information is dealt with by discarding portions of the plan which are dependent on the new information. Replanning fills in the rest in a manner appropriate to the new situation.

MEND does not specifically deal with the problem of surprises, and will remain oblivious to new information about the world as long as it does not cause an immediate failure. It deals with the situation when failures occur, but is short-sighted in not checking for future failures.

Sacerdoti describes a system called NOAH [Sacerdoti 75] which uses a data structure called a procedural net for planning and execution. NOAH does not have a good model of why actions fail and therefore resorts to a hierarchical search for an erroneously executed substep

when a failure is detected. Having identified the substep which failed, NOAH responds by replanning to achieve the intended effects of this action.

The approach that has been adopted in MEND is closest in spirit to that taken by Sussman and Goldstein. Sussman's system, HACKER [Sussman 73], is designed to learn to build structures in the BLOCKS world, and Goldstein deals with the problem of debugging incorrect "line drawing" programs through a system called MYCROFT [Goldstein 74]. The main difference is that their systems are addressed to the problem of handling conceptual errors. These conceptual errors arise either as a result of a lack of knowledge about the domain or because of an erroneous first attempt at planning, where interactions between related steps are not considered. In spite of this basic difference, we can make a point of comparison with HACKER. HACKER's analysis of failure reasons is based on an explicit representation of the previous states of the world in different contexts and an implicit representation of previous actions in the control frames created by calls on the action routines. Such mechanisms are quite impractical. In analyzing failures, MEND does not have access to the previous states of the world, and can only use currently available information and a record of actions that have been executed.

1.4 AN OVERVIEW OF THE REPORT

The following two chapters describe the hardware and software of the JPL robot, the structure of MEND, the world model, the interpretation of actions, and the detection of failures. Chapter 4 describes the process of failure reason analysis and how it is used in recovery from failures. A number of scenarios illustrate MEND's capabilities. Chapter 5 presents a brief outline of multiple outcome analysis and discusses the problem of integrating failure reason analysis and multiple outcome analysis. The final chapter summarizes the results, discusses the limitations of MEND, and suggests possible extensions.

THE JPL ROBOT

2.1 INTRODUCTION

The problem of error recovery discussed in the previous chapter was studied as part of the Jet Propulsion Laboratory's research program in robotics. In fact the results reported in this thesis are abstractions of the design concepts implemented in a module called MEND which is one of the components of the robot software system. This thesis gained substance from some of the actual problems encountered in the JPL robot system, and these problems placed realistic requirements on the program designed for error recovery.

This chapter gives an overview of the JPL robotics program, describing its motivation, hardware, tasks and goals for the robot, the components of the system, and their structural relationships.

2.2 JPL PROGRAM OBJECTIVES

Robots, autonomous or semi-autonomous machines with human like capability, are considered essential for planetary exploration. Machines that perform tasks on a step by step basis under human control are very inefficient when there is significant time delay in communication between the human control center and the machine. Difficulties are further exaggerated because of limitations in channel capacity. Because conditions in the territory to be explored are either

largely unknown, or at least unpredictable in detail, manipulation and locomotion will have to be achieved by execution of a large number of highly conservative steps. It is only by such means that environmental hazards can be avoided with reasonable chance of success. The above considerations motivate the JPL robotics research program. The long range goals of the project are to demonstrate the usefulness of artificial intelligence concepts for integrated robot systems, and arrive at guidelines for their design. More specifically, the immediate goals of the program are to build a breadboard robot system. This will be described next.

2.3 JPL ROBOT HARDWARE

The hardware of the breadboard robot system is shown in Figure 1.1. The manipulator, TV cameras and the laser rangefinder are mounted on a four wheeled vehicle. The hardware thus provides for three essential functional capabilities -- manipulation, vision and locomotion.

2.3.1 ARM hardware

The manipulator is a modified version of the Scheinman arm and is described by Dobrotin[73]. Figure 2.1 shows the kinematic configuration of the JPL manipulator. It has six degrees of freedom which allow the hand to be placed in an arbitrary position and orientation. The hand is a parallel jaw mechanism and has touch sensors mounted on the inside of the fingers. The positions that can be reached define the work space of

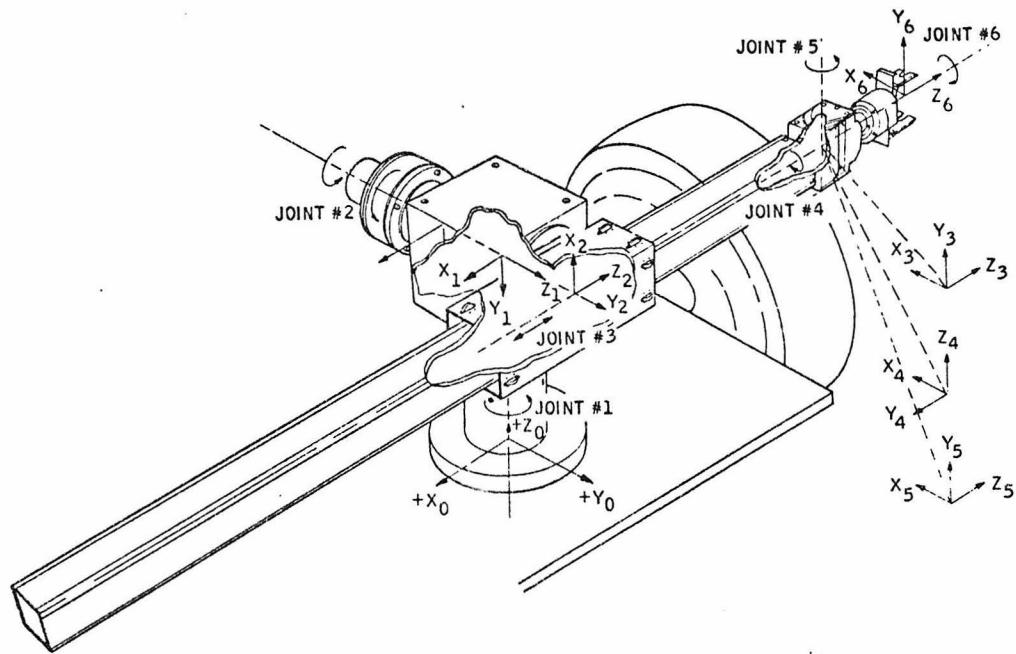


FIGURE 2.1

the manipulator. A detailed study of the work space of the JPL manipulator has been performed by Bejczy[72]. The manipulator can lift objects of about five pounds, and fast system response makes it possible for most trajectories to be executed within five seconds.

The manipulator is driven by permanent magnet torque motors. Electro-mechanical brakes hold the manipulator in position without the need for driving the motors. Feedback information is made available from each joint by means of potentiometers for position information, and tachometers for rate information. These are used by the servo loops implementing the real time control of the manipulator. Subsequent chapters will show the importance of position feedback information for purposes of error recovery.

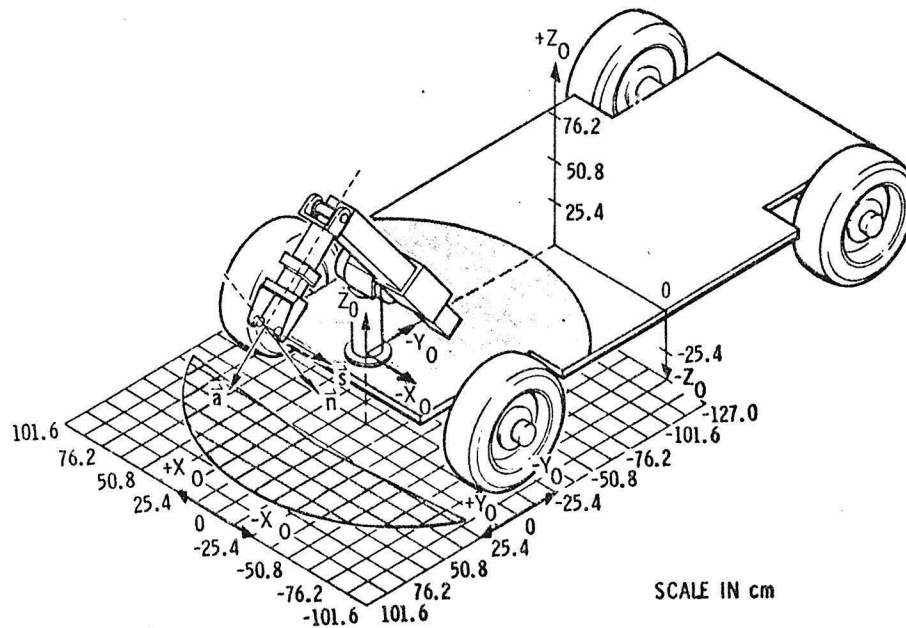


FIGURE 2.2

Figure 2.2 shows the reference frame which is used for manipulation purposes. This reference frame will be called the robot coordinate system. The actual position of the hand will be referred to as the position in robot coordinates. The position of the manipulator can be computed from the joint angles as measured by the potentiometers. The equations defining this transformation are given by Lewis[74]. Let us call the position computed in this manner, the position in hand coordinates. In general the position in robot coordinates will be different from the position in hand coordinates because of potentiometer errors, digitization errors and conversion errors. This error will be called the robot-hand position error, and as we shall see is the cause of several kinds of errors.

2.3.2 EYE hardware

The combination of the TV camera and the laser rangefinder system will be collectively referred to as the vision system [Williams 76]. Both are mounted on a pan and tilt mechanism and provide the primary means of sensing the environment.

Two solid state (charge-injection device) TV cameras provide stereo image data to the robot. The image array has 244 lines with 188 elements per line. The laser rangefinder is built around a gallium arsenide pulsed laser. The resolution in the workspace of the robot is about a quarter inch in the horizontal direction and a half inch in the vertical direction.

As with the manipulator, there is a position error that is to be contended with in the vision system. The coordinates of a point as determined by the vision system will be called the position in eye coordinates. These are the coordinates that will be used for positioning the manipulator in a desired location. The robot-eye position error can cause certain failures in manipulation, and these will be discussed in later chapters.

2.3.3 VEHICLE hardware

The vehicle provides for the mobility of the robot. The four wheels are independently driven by DC torque motors. The front and rear wheels can be steered independently by an Ackerman type double steering. The load capacity of the vehicle is about 500 pounds and travel speed is

limited to about 1 mile/hour. Position feedback is available through odometers. Tachometers provide velocity information and a directional gyro compass provides directional reference.

The vehicle has only recently been integrated into the system. It has not been considered in detail in the investigation reported here.

2.4 TASKS AND GOALS

An extensive study has been performed delineating science requirements for a Mars roving mission [Choate 72]. The science requirements of such a mission provide goals for an investigation such as the one undertaken in the JPL robotics research program. The breadboard system, however, is limited to a much narrower set of immediately realizable goals.

Consider sample collection. The problem of deciding what rocks are of scientific interest is beyond the capabilities of the system, and is likely to remain so for some time to come. These decisions are to be made by a team of science experts, who will monitor the activities of the robot and make selections of samples to be collected and decide what experiments are to be conducted, etc. To give a feeling for the interactions involved in such a semiautonomous system we describe a hypothetical scenario.

Suppose that the robot is commanded to pick up a specified rock and to put it on a viewing table. To achieve this the robot will have to use the vision system to build a three dimensional model of its environment and in particular to determine the location of the rock. Deciding on a grasping orientation, the robot will then compute a trajectory for moving the hand into a position appropriate for grasping the object. After picking it up, it will then compute a second trajectory to place the rock on the viewing table. We can imagine that the viewing table makes it convenient for examining the rock more closely, in order to make a decision whether or not it is worth keeping. Such decisions are likely to be time consuming and in the meantime the robot can be commanded to do other tasks. Perhaps a rock can be removed from an experiment chamber and transported to a sample container. Once the decision has been made about the rock on the viewing table, the robot can then be commanded, for instance, to discard the rock.

We note that some interactions are essential for scientific reasons. However, we expect the robot to make many operational decisions about when to use vision, or how to grasp a rock, on its own. It is in this connection that error recovery becomes important since we would like to minimize operational interactions. The reasons for this have been discussed in detail in several studies[Hooke 74, Whitney 74], and from a pragmatic point of view, these reasons provide a major motivation for systems with automated error recovery.

2.5 SYSTEM COMPONENTS AND STRUCTURAL ORGANIZATION

From a software point of view, one of the major problems in building a robot is the integration of manipulation, locomotion and vision. The JPL robot system embodies one approach towards this problem. The rationale and design considerations have been described in detail by Weinstein[75a, 75b].

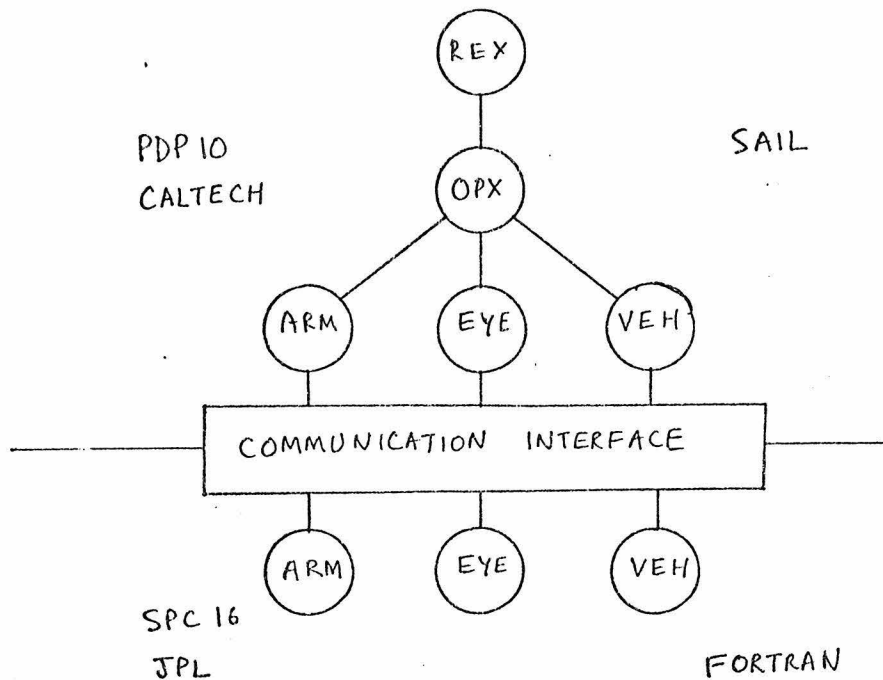


FIGURE 2.3

The functional capabilities of the robot hardware are incorporated within three software modules for manipulation, locomotion and vision. These modules are called ARM, ROVER and EYE respectively, and are collectively referred to as operative modules. The operative modules are controlled by the OPERative eXecutive (OPX), which in turn receives commands from the Robot EXecutive (REX). These five modules

are separate concurrent processes, hierarchically structured as shown in the Figure 2.3 [Srinivas 73a, 73b, Stevens 74, Roth 75].

Figure 2.3 shows the actual division of the modules between two different computer installations. The software on the PDP 10 is largely written in SAIL [VanLehn 73], while the software for the SPC 16 at JPL is written in FORTRAN. Each of the modules on the PDP 10 is implemented as a separate job. Communication between jobs is achieved through a message passing mechanism called MAILER.

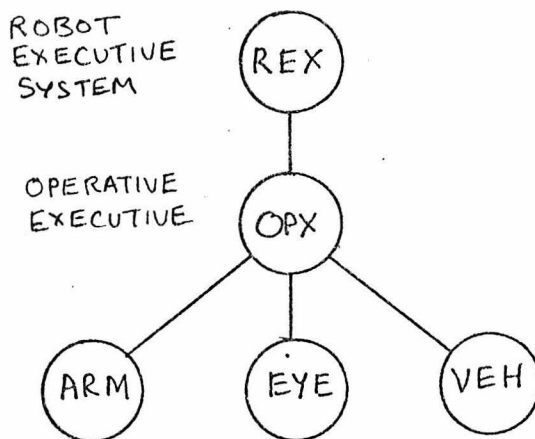


FIGURE 2.4

Note the division of the operative modules into two components. The details of Figure 2.3 will be largely ignored in the subsequent sections of this report, and the more abstract structure of Figure 2.4 will be used. The components of this figure will be described in functional terms in the following subsections.

2.5.1 Robot Executive System (REX)

REX serves as the interface between the human supervisor and the rest of the robot software. Such an interface is necessary because the robot is not a completely autonomous system, and ultimate control resides in the human supervisor. The supervisor is thus provided with a set of commands to control and monitor the activities of the integrated system. One of the design requirements is that REX be almost immediately responsive to the human supervisor. It is for this among other reasons that REX is a separate process running concurrently with the rest of the robot software.

Through REX the human supervisor can interact with the robot in the following ways. He can:

- (1) Create, edit and delete plans. A plan editor is incorporated as part of REX.
- (2) Request execution of plans and abort plan execution.
- (3) Determine the status of the system to many levels of detail.
- (4) Trace the flow of control and data between the different modules.

2.5.2 The Operative Executive (OPX)

The operative modules are individually responsible for the specific functional capabilities of the robot. To achieve a collective effort these modules are integrated by means of OPX. OPX achieves this by invoking primitive actions implemented in the operative modules in an order specified in a plan. Thus the primary function of OPX is the execution of plans.

Plans are structured sets of action units or other plans. The plan language allows for concurrent execution of action units in the different operative modules through a COBEGIN-COEND feature. However, we will restrict plans to sequential list of actions in this study. Each action unit specifies the operative module to be invoked and the specific action routine to be called. OPX sends to the appropriate operative module a message requesting the execution of a primitive action. On completion of the action, OPX continues with plan execution by repeating the above process with the next action unit. When the plan has been successfully executed a completion message is sent up to REX.

During execution of the plan OPX is still responsive to messages from REX. OPX can therefore determine the status of the operative module, abort the plan, or turn on tracing, so that the flow of control becomes explicit to the user.

2.5.3 The Operative Modules (ARM, EYE, and ROVER)

The operative modules implement primitive action capabilities of the robot. The implementation is termed an action unit, and a collection of such action units defines the primitives on which more complex plans are based.

The implementation of these primitive actions is a difficult task in itself. As example we consider some action units in each of the operative modules.

(1) ARM : MOVETO

The MOVETO action unit takes the manipulator from its current location to the specified goal position. In doing so it has to find a trajectory such that collisions are avoided. The manipulation system implemented by Lewis[74] includes a simple collision detector, but the system is likely to show markedly improved capability with the integration of the software developed by Udupa[76]. Udupa's solution is based on a theoretical framework which makes the task of collision detection and avoidance computationally tractable.

Once the trajectory has been determined, the joint angles need to be determined as a function of time, so that the manipulator will trace the desired trajectory. This again is a fairly complex task involving both kinematics and dynamics of the manipulator. Lewis[76] has implemented a very elegant solution to this problem.

(2) EYE : LOCATE

To LOCATE an object, the vision system needs to segment the digitized TV picture. Once the picture has been segmented the scene is analyzed, and a three dimensional model of the scene is built up using a combination of laser and stereoscopic information. Both segmentation and building 3-D models are difficult tasks involving considerable computational power. Williams[76], who has presented an overview of the JPL robot vision system, discusses these and other problems.

(3) Rover : GOTO

For the robot to be mobile, a path planning algorithm [Udupa 74, Thompson 75] is necessary. Given a description of the environment, the path planning algorithm finds a safe route to the goal location. The GOTO action unit then takes the vehicle along the planned route. The real time control of the vehicle is an extremely difficult problem especially in uneven and hazardous terrains. A first version [Thompson 76] of such a real time controller and a path planner is now capable of moving the vehicle around in a relatively flat environment in which rocks of various sizes are strewn around.

2.6 EXAMPLE OF EXECUTION

Consider a simple plan called PICKUP with the following structure:

```
(rock)
BEGIN
LOCATE (rock);
MOVE-HAND-TO-GRASP (rock);
GRASP (rock)
END
```

(We will use the convention that identifiers in lower case letters are uninstantiated parameters.) The human supervisor starts things rolling by commanding REX to execute the plan PICKUP with the parameter ROCK1. The plan is instantiated (with ROCK1 for "rock") and then sent to OPX for execution.

OPX begins execution by invoking vision to LOCATE ROCK1. This results in a TV picture being taken. The picture is digitized and then segmented. ROCK1 is identified in the segmented image and the vision system then builds a three-dimensional model of the rock. A descriptor of ROCK1 is returned which contains information about its location, width, height, support level, etc.

On receiving a completion message from the EYE operative module, OPX sends the next action unit to the ARM module. With the newly updated data for ROCK1, the ARM module determines an orientation for positioning the hand around the object. Trajectory computation is performed and the ARM executes the trajectory. A completion message is then sent to OPX. OPX continues execution by sending the GRASP action unit to the ARM module. The ARM module executes GRASP. The completion message from this action unit signals plan completion. OPX sends a message to REX with this information.

2.7 ERRORS AND A MODULE FOR ERROR RECOVERY

The execution of the plan PICKUP(ROCK1) described in the previous section assumes that every action succeeded in achieving its goal. Consider a case in which the hand bumps onto the object, instead of being correctly positioned around it. On detecting such an error, the ARM module sends an error message to OPX. OPX terminates plan execution by aborting the plan. An error message is sent up to REX, and the human supervisor is given the responsibility of planning recovery.

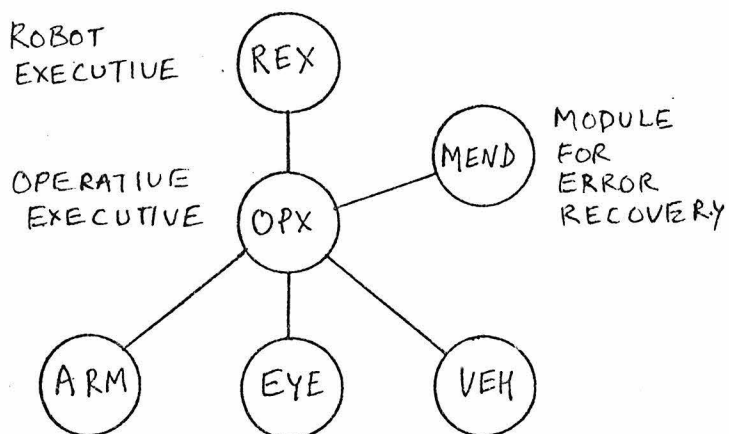


FIGURE 2.5

The goal of this thesis is to automate recovery from such failures. For this purpose a module called MEND has been implemented and its place in the robot structure is shown in Figure 2.5. The following chapters will describe MEND's capabilities and its internal structure. To illustrate the techniques used by MEND in planning recovery from failures, several scenarios dealing with the Hand-Eye subsystem of the JPL robot will be described.

A DESCRIPTION OF MEND

3.1 INTRODUCTION

In the previous chapter the need for a recovery system like MEND was explained. The relationship of MEND with other modules of the JPL robot system was briefly described. In this chapter we take a look at its internal structure. This chapter also describes the representation of knowledge about actions, the world model, the interpretation and execution of actions, updates to the world model and other details of MEND.

In its initial conception, MEND was expected to come into play only when a failure was detected. We can depict such a relationship as shown in Figure 3.1.

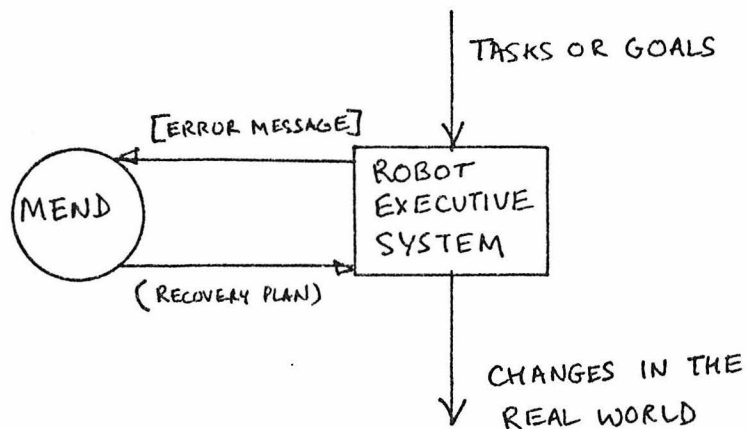


FIGURE 3.1

Tasks or goals are specified to the robot executive system which translates these into actual changes in the real world. If a failure is detected during execution of the task, then MEND receives an error message describing the failure. MEND responds to this by analyzing the failure and suggesting appropriate corrective steps. These corrective steps are structured into a recovery plan which is then sent over to the executive system.

The first version of MEND that was implemented reflected this initial idea. Its limitations provided a major motivation of the system as it exists now. Among other things it became clear that there was a need for a much more intimate relationship between the execution of plans and the analysis of failures. The second version of MEND plays a much more important role in the system. The description in this report is a conceptualization of this second version.

3.2 THE INTERNAL STRUCTURE OF MEND

MEND consists of many parts which interact together to execute plans, recover from failures etc. Every computation in MEND has access to three entities and these are shown in Figure 3.2. Knowledge about actions is built into the system and is not modified by any computation. The world model represents the current state of the world and the robot. The execution of actions results in updates to the world model to reflect the changes in the real world. The execution trace represents information about the plan currently in execution, such as what action

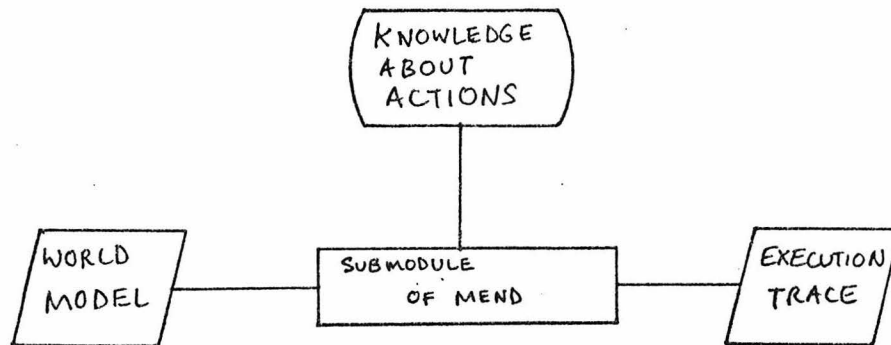


FIGURE 3.2

is to be executed next, what previous action failed, what corrective actions were taken, etc. These entities are discussed in more detail in later sections.

3.2.1 Execution of plans with MEND

At first we ignore the problem of error recovery and merely consider the execution of plans. Figure 3.3 indicates the flow of control between the different computations in MEND, and the two major states that MEND can be in when quiescent.

Some notational conventions are first described. Circles represent states and rectangles represent computational processes. Transitions from a state are activated when a message is received. Messages received by MEND are enclosed in square brackets with the sender being identified as a prefix to the message. A process can have several outcomes and can lead to other states or processes. Typically when a process leads to a state, a message is sent to another module (OPX).

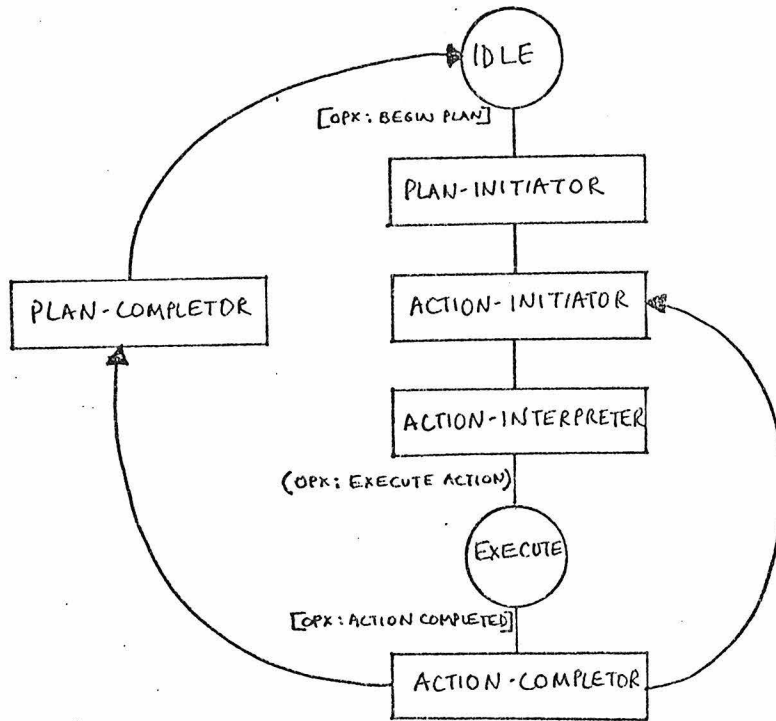


FIGURE 3.3.

These messages are enclosed in circular brackets, the receiver being identified as a prefix to the message.

With these conventions in mind, we can easily interpret Figure 3.3 in the following manner : If MEND receives a message indicating that plan execution is to be initiated, it readies itself for plan execution. PLAN-INITIATOR builds a structure representing the plan to be executed and initializes the status by pointing to the first action in the plan. It then transfers control to ACTION-INITIATOR. ACTION-INITIATOR checks preconditions to verify that the action can in fact be executed. Then the action is interpreted by ACTION-INTERPRETER in terms of simpler action primitives which are directly executable by the operative system. A message requesting execution of the primitive is sent to OPX and MEND

goes into the EXECUTE state. When MEND receives a completion message, ACTION-COMPLETOR checks that the results are as expected, updates the world model, updates the plan status and then transfers control to ACTION-INITIATOR if there are more actions to be executed. Typically a cycle of ACTION-INITIATOR's and ACTION-COMPLETOR's will sequence through the plan until execution of the plan is completed. When this occurs, PLAN-COMPLETOR cleans up the execution trace and returns MEND to the IDLE state.

3.2.2 MEND with error recovery

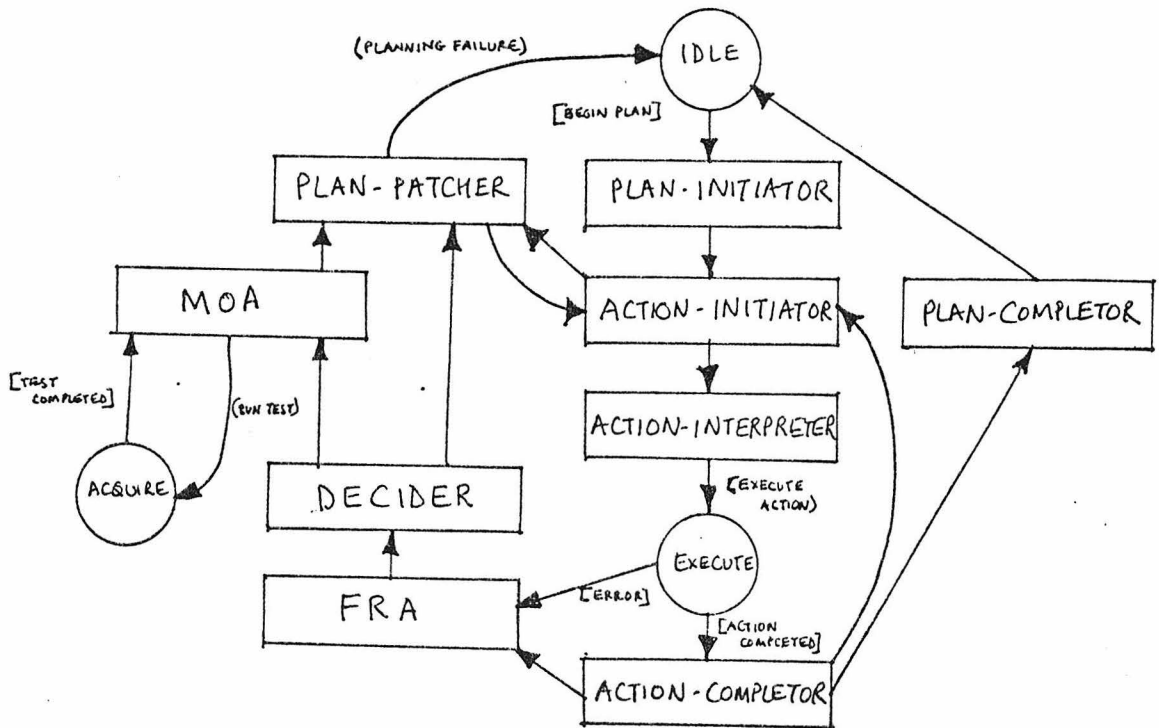


FIGURE 3.4

MEND, as represented in Figure 3.3, monitors the execution of plans by checking preconditions before executing actions and by checking postconditions to ensure that actions have been performed correctly. The figure does not indicate what MEND does when these checks fail or if an error message is generated by an operative module as a result of executing an action. Figure 3.4 answers this issue.

We can immediately note from Figure 3.4 that there are two ways in which failures are detected and control can be transferred to FRA, the failure reason analyzer. The path from the state EXECUTE to FRA represents those situations where failure is detected by the primitive actions. It is not always possible to rely on the primitives to detect the failure since failure is meaningful only in the context of the specific way in which the primitive is being used. These failures are detected by ACTION-COMPLETOR which has knowledge about the post-conditions of the actions interpreted by MEND.

The checking of preconditions, which is done as part of the ACTION-INITIATOR, is a way of anticipating and avoiding failures during execution. If the preconditions are found to be false, the PLAN-PATCHER is called to achieve these preconditions. If they have been only partially verified then these are marked as being UNVERIFIED in the execution trace, and execution is allowed to continue. (This will be explained in greater detail.)

FRA, DECIDER, MOA and PLAN-PATCHER perform the computations responsible for handling recovery from failures. FRA analyzes the failure and finds one or more explanations for the failure. DECIDER

provides a means of integrating failure reason analysis with multiple outcome analysis. In cases where the failure reason analysis has provided a simple solution for planning recovery, DECIDER transfers control to the PLAN-PATCHER. Otherwise, MOA is called. In analyzing multiple outcomes MOA may find that the available information is inadequate. In such circumstances MEND resorts to model-driven information gathering by asking for execution of simple actions, and going into an ACQUIRE state. On completion of the information gathering step, multiple outcome analysis is continued. When analysis is complete the PLAN-PATCHER is called with the results of the analysis. PLAN-PATCHER uses the results of the analysis to find a set of corrective steps and patch the plan in execution.

3.3 WORLD MODEL

The state of the world is represented in MEND through a set of data structures that will be collectively referred to as the world model. Typically, other systems model the world in terms of assertions represented as lists of items. MEND's representation is much more structured and tailored around the entities that it needs to deal with. The advantage of this is ease of accessing and efficiency of implementation, with the associated disadvantage of non-uniform procedures for accessing and updating the data base. Accessing of the world model occurs in several ways. ACTION-INTERPRETER needs to use information about the location and orientation of objects in interpreting actions in terms of simpler primitives. An

INTERPRET-ROUTINE associated with actions is programmed to access the world model to determine such information. In other words, knowledge about the way in which data are represented in the world model is built into these procedures.

Another set of procedures that access the world model are the INITIATE-ROUTINE's. These routines check preconditions and constraints applicable to the action with which they are associated. In MEND checking of preconditions and constraints is somewhat different from traditional robot problem solving systems. We illustrate with an example. Consider the precondition WITHIN-GRASP(ROCK) for GRASP(ROCK). The INITIATE-ROUTINE for GRASP will check that the hand position is within a small tolerance limit of the known location of the ROCK. If this check fails, then the precondition will be considered to be false. However if this condition is found to be true, MEND recognizes that this precondition may still not be true because of possible inaccurate information about the location of the ROCK. If in addition, though, MEND finds that the touch sensors in the hand are activated, then it will consider WITHIN-GRASP to be true. The main point to be noted is that precondition testing is not equivalent to looking for an appropriate assertion in the world model, or to deducing this from other assertions.

A third set of procedures, the FINISH-UP-ROUTINES associated with each action, update the world model and check completion conditions.

There are four entities about which information needs to be represented, and they are the state of the hand, objects, stations and frames. We discuss each of these in turn.

3.3.1 The hand state descriptor

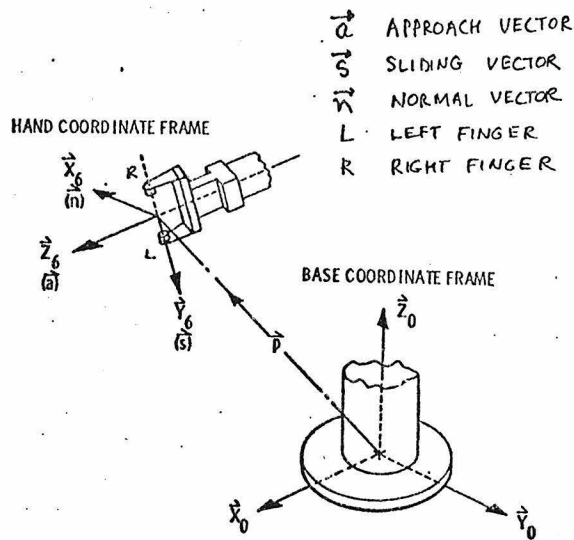


FIGURE 3.5

The hand state descriptor is updated after execution of any ARM action. The values of all the joint angles of the manipulator are recorded as attributes of the hand descriptor. From these joint angles it is possible to calculate the position of the hand, the sliding vector, and the approach vector in the Cartesian robot coordinate system [Figures 2.2, 3.5], but these are redundantly represented in the hand state descriptor. (The sliding vector refers to the vector along which the fingers move when the hand is opened or closed. The approach vector points in the direction of the fingers. Both these are indicated in Figure 3.5.) The hand state descriptor also includes information about the state of the touch sensors, and about the finger separation.

In addition to the above data, additional information is stored as part of the hand state descriptor. An attribute called GRASP-STATE indicates whether the hand is holding an object, touching an object, or whether the hand is empty. Another attribute, the GRASP-HEIGHT records the z coordinate of the hand whenever an object is grasped. This piece of information is useful in determining how far above the station the hand should be when attempting to place the grasped object at a desired location.

3.3.2 Object descriptors

Each object is represented as an item in the data base, and is described by various attributes. The vision system estimates the location of the object from a stereo view of the scene with additional information from the laser system in some cases. It also estimates an orientation of the object (the axis of the longest dimension), the grasping direction, the width along the grasp direction, and the height of the object, the support level, and other such details. All these are represented as part of the description of the object. In addition to these an attribute called ABSTRACT-LOCATION indicates whether the object is at a station, in the robot's hand, or whether the fingers are merely touching the object. This information is useful in testing pre-conditions and also in interpreting the attributes of an object in a meaningful manner. For instance, if the object has been grasped, then the location of the object is determined from the hand position and not from its LOCATION attribute. A point to note about the representation in MEND is that there is redundancy in the the world model. If the

ABSTRACT-LOCATION indicated that the object is in the robot's hand, the GRASP-STATE of the hand will also indicate that the robot is holding the object. This redundant description makes it convenient to get the necessary information directly rather than by an associative or pattern directed search.

3.3.3 Station descriptors

Stations provide a means of naming some designated locations. These locations can either be on the ground or on the robot platform. Station OCCUPANCY indicates whether or not there is an object at the station. This will be checked, for instance, before an object is transported to the station to be placed there. A STATION-TOLERANCE-LIMIT can be specified and later used in determining whether a placement of the object at the station is within acceptable limits.

3.3.4 Frame descriptor

A frame defines the position, sliding vector and approach vector of the hand, and is useful in specifying intermediate configurations when the hand moves from one location to another. With a good obstacle avoider built into the manipulation system it should be possible to eliminate this low level of dealing with hand configurations. However, they are necessary in the current system to avoid obstacles.

3.4 KNOWLEDGE ABOUT ACTIONS

The implementation of ARM, EYE, and VEHICLE provide higher levels of the robot system with action primitives. These primitives can be used to achieve various results. For instance, the primitive MOVETO which can move the hand from one position to another, can be used to position the hand around the object or to transport the object from one position to another. In the first case, it would be reasonable to check that the hand is empty before beginning execution of the action, while in the latter case it is necessary to verify that the object to be transported is in the hand. Modelling of actions directly in terms of these primitives is difficult because of the different conditions that apply in different circumstances.

MEND thus models a slightly higher level set of actions which are specialized for different functions. MOVE-HAND-TO-GRASP(object), for example, is a specialization of MOVETO with the implicit purpose of positioning the hand around the object to be grasped. When the human specifies a task by writing a plan composed of these higher level actions there is an implicit notion of its intended effects. It is this notion of purpose or intent that makes it possible to recover from failures in actions in a reasonable manner.

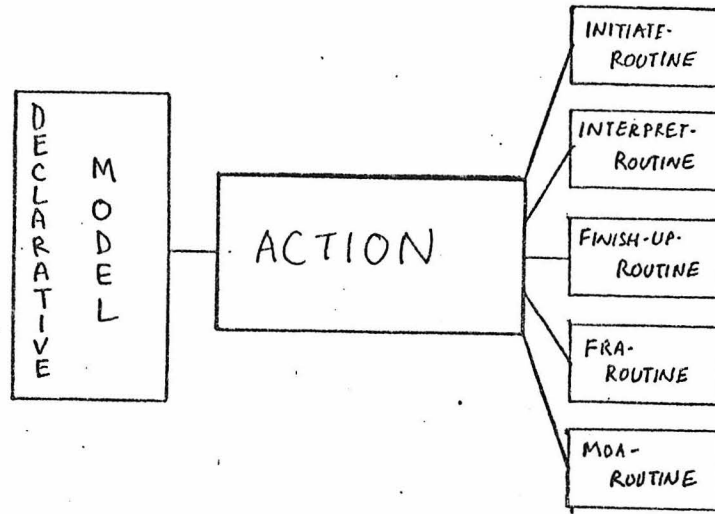


FIGURE 3.6

Each action is represented in MEND as an item in the data base, and has both a declarative and procedural component[Figure 3.6]. The declarative model is represented as triples in an associative data base[LEAP data structures in SAIL - VanLehn 73], and a triple (A I V) is interpreted to mean that attribute A of item I has value V. A partial description of the declarative model of MOVE-HAND-TO-GRASP is shown below:

- (ACTION-TYPE MOVE-HAND-TO-GRASP (ROCK) EFFECTOR)
- (PRECONDITION MOVE-HAND-TO-GRASP (ROCK) (EMPTY-HAND))
- (PRECONDITION MOVE-HAND-TO-GRASP (ROCK) (OPEN-HAND))
- (NEEDED MOVE-HAND-TO-GRASP (ROCK) LOCATION (ROCK))
- (RESULT MOVE-HAND-TO-GRASP (ROCK) WITHIN-GRASP (ROCK))

This model represents the fact the MOVE-HAND-TO-GRASP is an effector action, that it has a parameter called ROCK, that one of its pre-conditions is EMPTY-HAND, that the location of the ROCK is needed information, that its intended result is WITHIN-GRASP(ROCK), and so on. The procedural component consists of an INITIATE-ROUTINE, an INTERPRET-ROUTINE, FINISH-UP-ROUTINE, a FRA-ROUTINE, and a MOA-ROUTINE associated with the action [Figure 3.6]. ACTION-INITIATOR, for instance, calls the MOVE-HAND-TO-GRASP INITIATE-ROUTINE to check parameter types and preconditions.

Let us now take a brief look at the actions that have been modelled for demonstrating error recovery in simple manipulation tasks.

3.4.1 ARM actions

(1) MOVETO(frame)

This action defines the most primitive capability of the manipulator to move from one location to another. The parameter of MOVETO specifies the desired absolute position and orientation of the manipulator. This action is not directly modelled in MEND but the following specializations of this action are.

(1a) MOVE-HAND-TO-GRASP(object)

The expected result of this action is to position the hand such that the fingers surround the specified object. The initiation routine will check that the hand is empty, that the fingers are open, and that the object is graspable. The INTERPRET-ROUTINE

determines a position and orientation which is appropriate for grasping the object. In general this could be a difficult problem requiring a good understanding of the shapes of objects and the resulting constraints on how the object can be grasped. However, for the simplified world that MEND is being tested in, this is relatively straightforward. In fact, a grasp orientation is suggested by the vision system.

(1b) MOVE-HAND-TO(frame)

The only difference between this action and the primitive MOVETO is that the INITIATE-ROUTINE will check that there is nothing in the hand.

(1c) APPROACH(object)

Sometimes it is necessary to position the hand near the object so that the location of the object can be determined relative to the hand. This action positions the hand at a point some small distance above the object. The INITIATE-ROUTINE will check that the hand is not holding any object.

(1d) TRANSPORT(object, station)

An object which has been grasped can be transported to a station where it is to be placed. Preconditions that are checked ensure that the object is the hand, and that the station is unoccupied. The INTERPRET-ROUTINE will use the information recorded in the GRASP-HEIGHT attribute of the hand state descriptor in computing a

hand position which will result in the object gently bumping onto the desired station. The FINISH-UP-ROUTINE will check that the station tolerance limit is not exceeded.

(1e) MOVE-OBJECT-TO(object, frame)

This is similar to MOVE-HAND-TO except that in this case an object is expected to be in the hand.

(2) SEPARATE(finger!separation)

This action results in the fingers being opened (or closed) to the desired finger separation. This action again has several specializations.

(2a) GRASP(object)

Execution of this action results in the object being grasped provided that the fingers have been positioned around the object. The motors driving the fingers are kept active for a short period of time even after the object has been sensed by means of the touch sensors. This is done so that the hand gets a firm grip on the object. The INITIATE-ROUTINE will check that the position of the hand and the known position of the location of the object indicate that the object is WITHIN-GRASP. However, this test does not guarantee that the object is really within grasp because of possible errors in the location of the object. The FINISH-UP-ROUTINE will update the ABSTRACT-LOCATION of the object to indicate that the object is in the robot's hand. The

GRASP-STATE and GRASP==HEIGHT attributes of the hand descriptor are also updated.

(2b) CLOSE!UNTIL!TOUCH(object)

This is similar to GRASP. However the fingers stop moving the moment the touch sensors are activated. Often it is a good heuristic to gently close the fingers around the object and then squeeze tight in order to GRASP the object. After execution ABSTRACT-LOCATION will merely indicate that the hand is touching the object.

(2c) LETGO(object, station)

This merely SEPARATES the fingers to release the object. It verifies that there is an object in hand and that hand is at the station. The FINISH-UP-ROUTINE will change station OCCUPANCY to occupied, updates the ABSTRACT-LOCATION of the object to indicate that it is AT(station).

3.4.2 EYE action units

(1) LOCATE(object)

This action assumes that the object has been identified either by specifying an approximate location, or by certain distinguishing features of the object. A new descriptor is created by the FINISH-UP-ROUTINE and the different attributes updated with the information provided by the execution of this action.

(2) LOCATE-RELATIVE-TO-HAND(object)

On occasion it is necessary to determine the position of an object accurately. Part of the difficulty in determining the location accurately is the discrepancies between the hand coordinate system and the eye coordinate system. To avoid this problem the hand is brought close to the object, and the location of the object determined relative to the hand. As with locate, LOCATE-RELATIVE-TO-HAND updates data structures describing the object.

3.5 THE EXECUTION TRACE

A new execution trace is created by PLAN-INITIATOR whenever a new plan is to be executed. It starts off as a set of nodes strung together to represent the plan to be executed. Each node is associated with an action and its parameters. A NEXT-TO-BE-EXECUTED pointer indicates which action is to be executed next. An example of an initial structure is shown in Figure 3.7.

Several things get added to the trace as execution proceeds. Before beginning execution of an action, the INITIATE-ROUTINE will mark all unverified preconditions and constraints by creating triples of the form:

(UNVERIFIED NODE-C WITHIN-GRASP (ROCK))

(UNVERIFIED NODE-C GRASPABLE (ROCK))

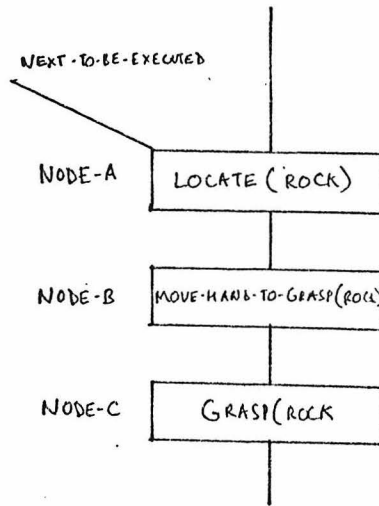


FIGURE 3.7

After successful execution, the NEXT-TO-BE-EXECUTED pointer is moved ahead to the next node. If failure occurs and PLAN-PATCHER successfully finds a recovery plan, it will modify the execution trace by creating new nodes for the action to be executed. We show an example for failure in NODE-B of Figure 3.7, corrected by a single step plan. The new structure is shown in Figure 3.8.

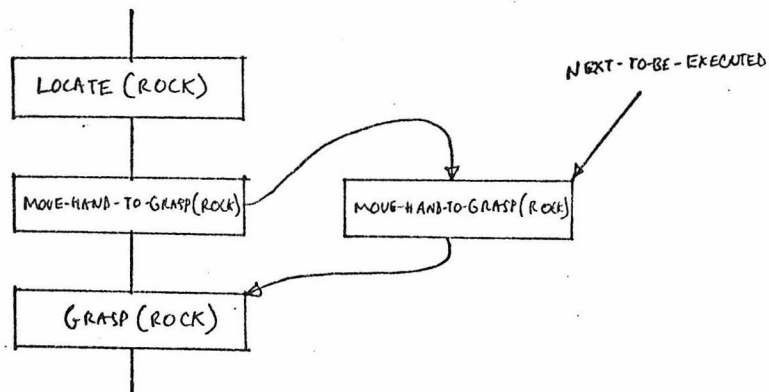


FIGURE 3.8

After execution of the recovery plan, execution will continue with the node following the failed action.

FAILURE REASON ANALYSIS

4.1 INTRODUCTION

MEND uses two different strategies for recovery from failures. In this chapter we discuss how MEND analyzes failure and how the results of this analysis are used in planning recovery. Let us first take a look at a classification of failure reasons.

4.2 A CLASSIFICATION OF FAILURE REASONS

MEND's analysis of failures reflects an understanding of four kinds of failures. These are operational errors, information errors, precondition errors, and constraint errors. We discuss each of these in turn.

Actions can fail to achieve their intended result because of certain inherent problems in executing the action. These errors are peculiar to the specific operation being performed and are therefore referred to as operational errors. We can see a number of examples of these kinds of errors in the JPL robot. For instance, the manipulator may deviate from the planned trajectory in moving from one location to another. This operational error of servoing will not often have any serious consequences, but in some cases it can cause the manipulator to bump into the object to be grasped.

The process of determining the location of objects is subject to several operational errors. Calibration, resolution limits, noisy data and other such reasons are all contributing factors in producing erroneous information. For our purposes there is no need to distinguish between them, and we will deal with them collectively as a single operational error. There is the possibility of confusion here, and we reiterate that the operational error does not refer to the inaccuracy in the location. This inaccuracy is a consequence of the operational error.

If the robot fails in positioning the hand correctly around the object because of inaccuracy in the location of the object, MEND will recognize this to be the result of an information error. We see that the operational error in locating an object has shown up as an information error when attempting to pick up the object.

Before executing an action the initiate routine will check preconditions and constraints. If any of these are found to be false the PLAN-PATCHER will attempt to make them true before continuing with the execution of the action. If all the preconditions of actions could be checked and verified, there would be no reason for failures to occur because of precondition errors. However, it would be unrealistic to expect a robot to check these preconditions exhaustively, since some of them are inherently difficult to check. Therefore, the execution monitoring in MEND does not require that all preconditions be absolutely verified before executing an action. In some cases executing subsequent steps may be the simplest way of finding out that they were or were not

satisfied. For instance, checking to see that an object is truly within grasp before actually grasping the object is unnecessary and expensive, since the very action of grasping will immediately indicate whether this is true or not. Even when previous actions were executed with the intentions of making these preconditions true, the robot cannot always depend on their being true since these previous actions may themselves fail for the various reasons under discussion. In any case, the consequence of not verifying preconditions is that after execution of an action, failure can be attributed to the falsity of preconditions. It is with this interpretation in mind that we talk of precondition errors. (Similar comments apply to constraint errors discussed later.)

From the discussion so far it may seem that information and precondition errors can both be traced back to operational errors. This is largely true but with a small qualification. We cannot expect a robot to keep a record of its activities (the execution trace) that is infinitely long. Therefore it is not always possible to trace back the reason for failure to an operational error. Also, the plan may have been produced with certain conditions being assumed to be true of the initial state of the world. In both these circumstances MEND recognizes some special cases : initial information error and initial condition error.

Finally, an action can fail because there are some constraints that must be satisfied for successful execution. An example of this is that the object should be MOVABLE before the robot can transport it from one place to another. In a sense, these are merely special cases of

preconditions with the limitation that the robot has no way of making them true. Of course, what is a "constraint" for one robot may be merely a "precondition" for a robot with a larger set of capabilities. But lacking omnipotence any robot will have certain limitations and thus encounter situations that it can do very little about. Constraints, therefore, model certain limitations in the capability of the robot.

4.3 THE FAILURE TREE

The results of failure reason analysis are represented in a structure called the failure tree. We first describe this structure, before looking at how it is built.

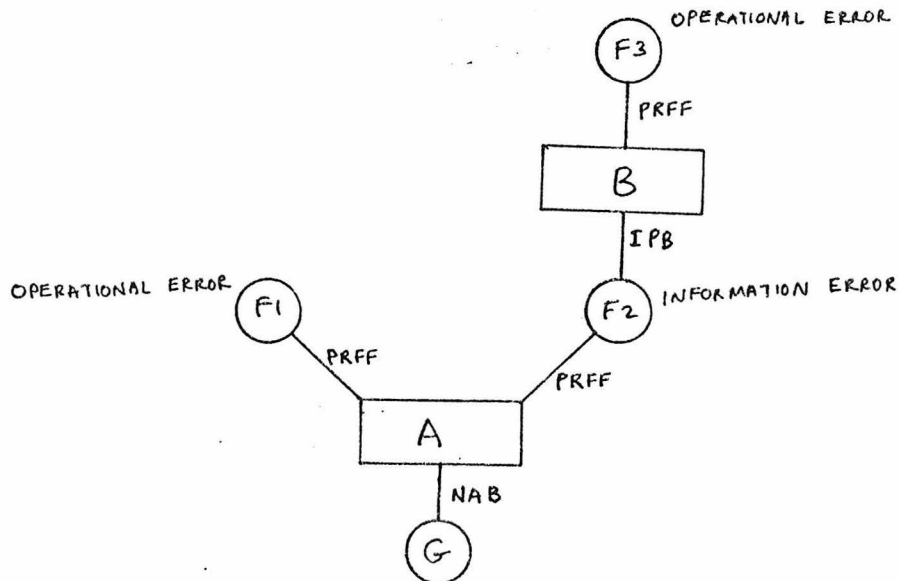


FIGURE 4.1

Figure 4.1 is an example of a failure tree. There are two types of nodes in a failure tree -- failure nodes and action nodes. Failure nodes are represented by circles and action nodes by rectangles. Action

nodes are linked to a node in the execution trace through a plan link which is not explicitly shown in the above figure. Each failure node identifies the failure and its type, and may point up to one or more actions nodes through one of several kinds of links. The NAB (Never Achieved By) and IPB (Incorrectly Provided By) are two examples of such links. Action nodes point up to failure nodes through a PRFF, the "possible reason for failure" link.

The example in Figure 4.1 can now be interpreted in the following manner : Goal G was never achieved by action A because of one (or more) of two possible reasons F1 and F2. F1 indicates that the failure could have been the result of an operational error, while F2 represents data incorrectly provided by action B.

The types of failure that are associated with failure nodes are the ones we have discussed earlier (operational, information, precondition, and constraint errors) with one addition. There is a special "failure type" which occurs only at the root of the tree and this represents a goal failure (GOAL-FAILURE). Operational and constraint failure nodes do not point up to anything since they are local to the failed action. Precondition nodes may link upward to action nodes that were intended to achieve the precondition. When such links are missing it means that no action was executed to achieve this condition. It therefore represents an initial condition error. Similarly, information failure nodes link upward to actions that provides the needed information, and missing links indicate initial information errors.

4.4 BUILDING A FAILURE TREE

The idea behind failure reason analysis is to find an explanation of the failure. By explanation we will mean a chain of reasons represented by a path from the root node of the failure tree to one of its leaf nodes (G - A - F2 - B - F3 in Figure 4.1). If we consider all possible reasons for failure at each action node, then the failure tree represents all possible explanations for the failure. Clearly, only one or a few will be relevant in any particular situation. We can now think of failure reason analysis as the process of limiting this set of all possible explanations. Of course, it will not always be possible to find a single explanation and MEND has to deal with several possible explanations in certain cases.

We next ask what can be done to constrain the set of explanations. Several things suggest themselves. Any unsatisfied precondition or constraint can be the cause of the failure. However if we know that the preconditions and/or constraints have been satisfied then the failure cannot be attributed to these causes. The execution trace in which unverified preconditions and constraints are noted down provide the information necessary for this purpose. We reiterate here that a precondition is not considered to be verified because a previous action was executed to achieve this, but rather is considered to be true when independent tests verify the truth of the precondition. To consider a trivial example, OPEN-HAND is verified by checking that the finger separation is equal to the maximum separation. Typically other robot systems verify preconditions by looking for explicit assertions in the

data base or by "deducing" the truth of these conditions from other assertions. The result of this is no distinction can be made between those preconditions that can be verified directly from feedback information and those that are only surmised to be true because a previous action was supposed to achieve this goal.

A second important way in which the failure nodes branching from an action can be cut down is by looking at the manifestation of the failure. Different reasons for failure manifest themselves in different ways, and by identifying distinctive features of the failure situation some of the reasons for failure can be shown to be impossible or at least unlikely.

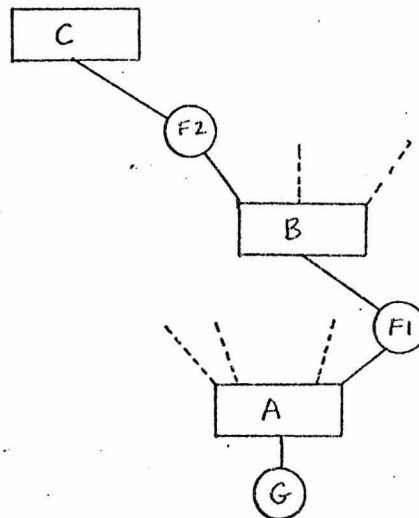


FIGURE 4.2

Thirdly, the history of actions can show that certain explanations are impossible. Consider, for instance the situation in Figure 4.2. Suppose that no reasons for failure of action C can be found. In that case we can conclude that B could not have failed because of F2. If

there are no other reasons for failure of B besides F2, we can further conclude that A could not have failed because of F1. Such reasoning can significantly cut down the set of possible explanations, thereby pointing out the real explanation of the failure.

In building a failure tree (i.e. in analyzing failures), MEND applies constraints at each action node, builds a tree, and finally cleans up the tree by eliminating the explanations that can be shown to be impossible on the basis of the history of actions. The following is an overview of the tree building algorithm:

1. Find the set of possible failure reasons.
2. Eliminate verified preconditions and constraint failure from this set by looking at the execution trace.
3. Look for distinctive features of the manifestation to further constrain the set of failure reasons.
4. For each of the remaining failure reasons create failure nodes and link the action node to these newly created failure nodes.
5. For precondition failure nodes find from the execution trace the previous action responsible for making the precondition true. For information failure nodes find the previous action which provides the necessary information. Create an action node for each of the previous actions so determined.
6. Repeat the above process for the newly generated action nodes.
7. Clean up the failure tree by eliminating impossible explanations (i.e. apply the process described in connection with Figure 4.2).

4.5 THE FAILURE REASON MODEL

We can now see that to execute the algorithm outlined in the previous section we need a model of actions that will provide the necessary information. This is precisely what the failure reason model is all about.

Firstly, the model should identify the set of possible failures. Operational failures are represented explicitly through a set of associations as shown below:

(OPERATIONAL-ERROR MOVE-HAND-TO-GRASP (ROCK) SERVO-ERROR)

(OPERATIONAL-ERROR MOVE-HAND-TO-GRASP (ROCK) COLLISION)

(OPERATIONAL-ERROR LOCATE (ROCK) INACCURACY)

Knowledge about other types of failures is implicit in the action model as presented earlier. Any precondition, constraint, or needed information is recognized as a possible reason for failure by MEND. Figure 4.3 below shows the correspondence between the action model and the failure tree.

- F1 : (OPERATIONAL-ERROR MOVE-HAND-TO-GRASP (ROCK) SERVO-ERROR)
- F2 : (OPERATIONAL-ERROR MOVE-HAND-TO-GRASP (ROCK) COLLISION)
- F3 : (NEEDED MOVE-HAND-TO-GRASP (ROCK) LOCATION (ROCK))
- F4 : (CONSTRAINT MOVE-HAND-TO-GRASP (ROCK) GRASPABLE (ROCK))
- F5 : (PRECONDITION MOVE-HAND-TO-GRASP (ROCK) EMPTY-HAND)
- F6 : (PRECONDITION MOVE-HAND-TO-GRASP (ROCK) OPEN-HAND)

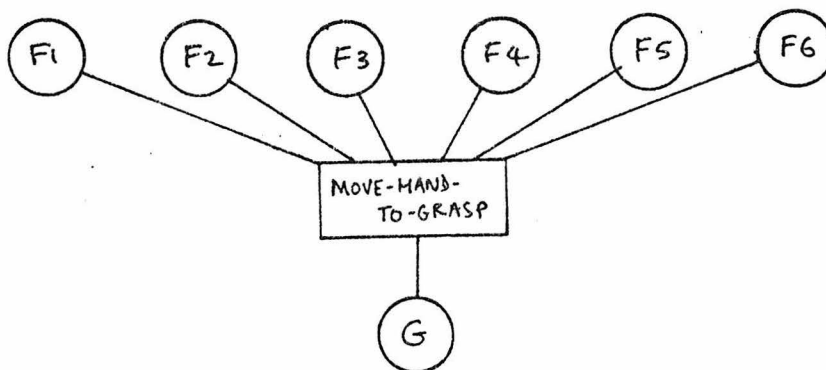


FIGURE 4.3

The knowledge necessary to eliminate some possibilities on the basis of their manifestation is highly domain dependent. This knowledge is incorporated in a procedure, the FRA-ROUTINE. The FRA-ROUTINE tests for the presence or absence of distinctive features of the failure in the current state of the world. These tests indicate that certain failure reasons are impossible, that others are likely, and so on.

To see what the FRA-ROUTINE is all about let us consider what information about the manifestation of the failure of MOVE-HAND-TO-GRASP can be used in constraining the set of failures. Figure 4.4 shows what the MOVE-HAND-TO-GRASP FRA-ROUTINE does. At each node in the tree, a condition is tested and the appropriate branch taken depending on the truth or falsity of the result. The leaf nodes of the tree specify a list of failure reasons that are considered possible in that specific context. The rationale for the results produced by the FRA-ROUTINE is not justified here, but is discussed when some scenarios are described in a subsequent section of this chapter.

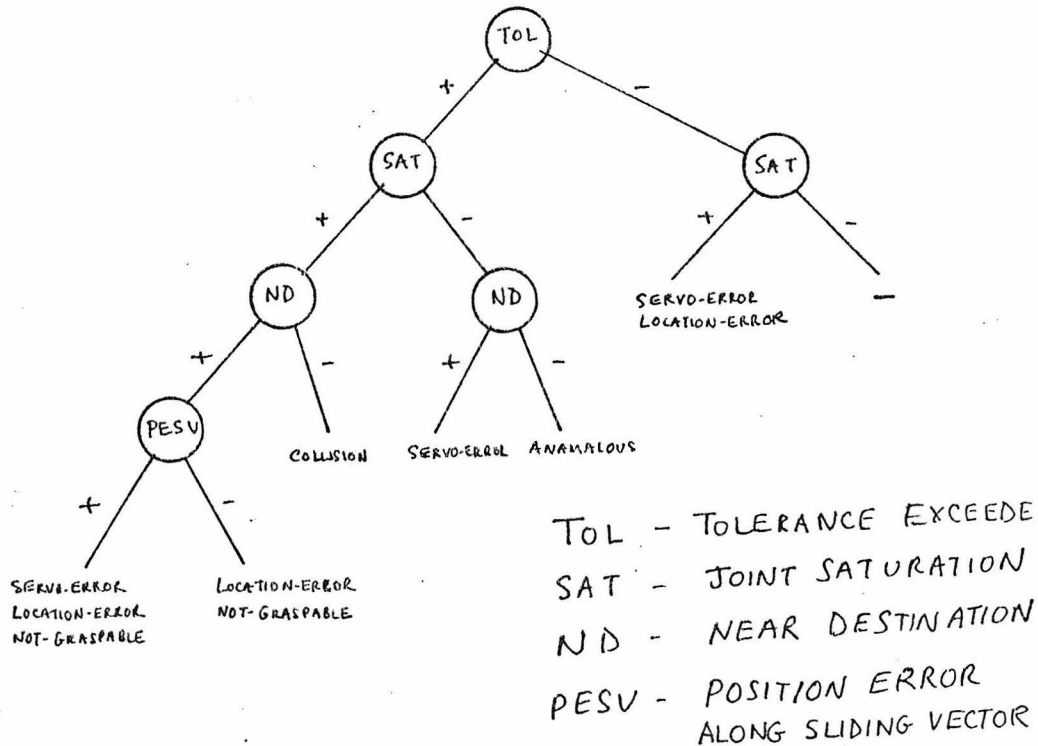


FIGURE 4.4

We note several important points about the FRA-ROUTINE. As with precondition testing, the conditions being tested are easily computed using the data in the world model and the feedback information. For instance, the condition NEAR-DESTINATION is checked by verifying that the final position of the hand is within a pre-specified limit of the planned destination. Similarly, PESV is checked by testing that the projection of the position error (the difference between the the actual position of the hand and the planned position) on the sliding vector exceeds a pre-set tolerance limit.

Secondly, note that the routine illustrated in Figure 4.4 assumes that the preconditions EMPTY-HAND and OPEN-HAND have been made true, and does not consider how failure resulting from falsity of these conditions will manifest itself. The reason that this is possible is that execution monitoring will catch these precondition errors and will not allow execution of this action to continue unless they have been satisfied. Also note that the FRA-ROUTINE represents a procedural incorporation of knowledge about the manifestations of failure.

It may seem unreasonable to make such assumptions or to embody such knowledge implicitly in procedures, when designing a general purpose system, but I believe it is essential to build in such simplifications when designing a practical system. It is perhaps appropriate to mention at this point that my philosophical viewpoint is that the flexibility of reasoning about actions from a declarative model can be carried only so far. A lot of discussion about declarative versus procedural representations is completely pointless since they are not considered in

the context of the goals of a well defined system. The decisions that were made in designing MEND are tailored to meet specific goals considered desirable for the JPL robot, with the consequence that MEND cannot "reason" about certain aspects of its knowledge base. In a different context, Martin[74] voices a similar viewpoint, and points out in connection with the MACSYMA system that "the implications to the system design of facts like the commutability of plus are so great that the system must be built assuming them to be true".

To get back to our discussion of the failure reason model. There is a problem in using the FRA-ROUTINE to limit the set of failure reasons in step 3 of the algorithm presented in the previous section. This is because this routine requires information about the state of the world in making its decisions. This means that previous states of the world need to be represented. To keep this information around would be quite unrealistic in any large system, even using a stack like mechanism for representing only the incremental changes.

There are two ways of tackling this problem. Execution monitoring can be extended by running the FRA-ROUTINE regardless of whether or not a failure occurred and marking the failure possibilities in the execution trace. Later, if failure reason analysis is necessitated in a subsequent action, this possibility list presents a "summary" useful for analysis.

A different strategy is to ignore the possibility of failure when there is no explicit triggering of the failure, and to later ask what failures could have escaped detection at the time of execution. This is answered by the representation of facts such as the one shown below,

(PTRIG (NEEDED MOVE-HAND-TO-GRASP (ROCK) LOCATION (ROCK)) NTRIG)

where PTRIG = POSSIBLE-TRIGGER

and NTRIG = NULL-TRIGGER,

which says that a location error in the position of the rock could go untriggered when executing MOVE-HAND-TO-GRASP. (The reason for this is that if the location error is sufficiently large, the hand will completely miss the object.) With these facts in hand, MEND can immediately determine the set of failure reasons to be considered in step 3 of the failure tree building algorithm, without having to run the FRA-ROUTINE for previously executed actions.

An assumption implicit in the latter approach is that any triggered failure has already been dealt with. The first approach is a more general technique but will perform unnecessary computations in situations where plans are successfully executed.

There is one other aspect of the failure reason model which has nothing to do with the analysis per se, but provides information about what

to do in case failure is attributed to a particular kind of failure. An example of this is shown below:

```
(TO-CORRECT (OP-ERROR MHTG (ROCK) SERVO-ERROR) MHTG-REC-PLAN1 (ROCK))
```

where OP-ERROR = OPERATIONAL ERROR

MHTG = MOVE-HAND-TO-GRASP,

```
and MHTG-REC-PLAN1 = (rock)
                     BEGIN
                     MOVE-HAND-TO-GRASP (rock)
                     END
```

This says that to correct an operational error of servoing in MOVE-HAND-TO-GRASP the plan MHTG-REC-PLAN1 should be executed. MHTG-REC-PLAN1 will attempt recovery by simple repositioning of the hand, and this is appropriate for servoing errors. In general, TO-CORRECT steps are provided for each operational error.

Constraint errors on the other hand have no such "imperative knowledge" [Goldstein 74] about what to do, since the robot does not have the capability to correct such failures. If failure is attributed to a constraint error, the plan is aborted and MEND seeks human aid.

Precondition and information errors are not directly associated with corrective steps and are treated somewhat differently than operational errors. Since operational errors are local to the action under consideration, all that is needed is knowledge about how to fix the situation. For precondition errors the obvious thing is to re achieve the failed precondition before trying the action again. But before this can be done, some of the effects of the failed action must

be undone. After the precondition has been achieved, other steps must be redone to get execution back on the right track. The PLAN-PATCHER in MEND implicitly incorporates this knowledge and recognizes that there are three parts to recovery from precondition errors : undo certain effects, reachieve precondition, and redo the undone steps. Knowledge about what needs to be undone and what needs to be redone is part of the failure reason model. For example,

```
(UNDO-STEPS (PRECONDITION GRASP (ROCK) WITHIN-GRASP (ROCK)) OPEN)
(REDO-STEPS (PRECONDITION GRASP (ROCK) WITHIN-GRASP (ROCK)) GRASP (ROCK))
```

says that if failure in GRASP is attributed to the precondition error of the object not being WITHIN-GRASP, the OPEN undoes the effect of GRASP, and that GRASPing the object needs to be redone after WITHIN-GRASP has been reachieved.

The situation is entirely similar for information errors, with the one difference that instead of reachieving the precondition, the intermediate step between undoing and redoing is that of getting the needed information.

Facts such as these are used by the PLAN-PATCHER. We will describe how this works when discussing the scenarios that illustrate MEND's capabilities.

4.6 SCENARIOS

In this section we look at several scenarios to illustrate how MEND copes with failures in simple manipulative tasks.

4.6.1 An operational failure

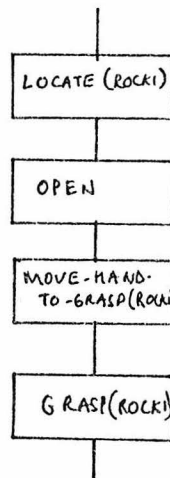


FIGURE 4.5

Consider the plan shown in Figure 4.5. On execution the action LOCATE updates the world model with the location, orientation, support level, and other details of ROCK1. The fingers are then opened in preparation for the next action of positioning the hand around the object. Preconditions are checked and finding them to be true MEND computes a grasping orientation and interprets MOVE-HAND-TO-GRASP in terms of the primitive MOVE-TO. The robot finds on execution of this action that tolerance criteria is not met and the primitive signals a failure by triggering TOLERANCE-EXCEEDED.

MEND immediately recognizes the possibilities shown in Figure 4.3. F5 and F6 are immediately eliminated from consideration since the preconditions were found to be true before execution. The FRA-ROUTINE associated with MOVE-HAND-TO-GRASP is then run. Since only a tolerance error was triggered this routine returns SERVO-ERROR (F1) as the only possibility. The FRA-ROUTINE eliminates COLLISION (F2) because there was no SATURATION, which is a second way in which the primitive MOVETO can trigger failure. (We are using saturation of the joint motors driving the manipulator as a sensor. Proximity sensors [Johnston 74] will more effectively provide information about such situations by detecting impending collisions.) LOCATION-ERROR (F3) and NOT-GRASPABLE (F4) are not impossible, but are considered to be unlikely in the absence of saturation. The rationale behind this is that a location error in the object to be grasped or largeness of the object (making it NOT-GRASPABLE) are likely to cause the hand to bump onto the rock, causing one of the joints to saturate.

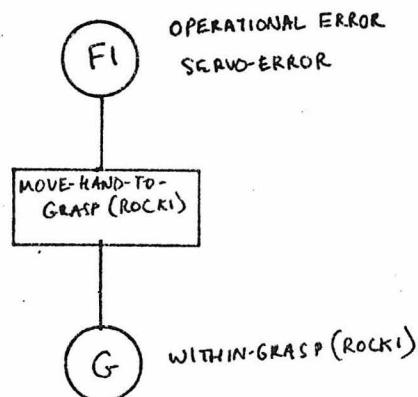


FIGURE 4.6

In any case, the result of running the FRA-ROUTINE is that the failure tree is pruned to the simpler structure shown in Figure 4.6. Since the error is an operational error, the PLAN-PATCHER looks for the TO-CORRECT plan associated with SERVO-ERROR in the failure reason model of MOVE-HAND-TO-GRASP. The corrective step is patched in and the modified plan is shown in Figure 4.7. MOVE-HAND-TO-GRASP will be interpreted in exactly the same way as before, with the result that the corrective step is equivalent to repositioning the hand.

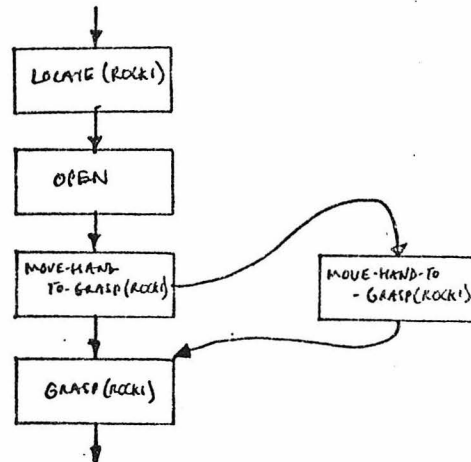


FIGURE 4.7

4.6.2 An information error

Consider a small change to the previous scenario and assume that the hand actually bumped onto the rock. Also assume that the rock had been previously picked up indicating that it is both GRASPABLE and MOVABLE. This time failure is triggered by two conditions -- SATURATION and TOLERANCE-EXCEEDED. SATURATION indicates that one of the motors driving the manipulator saturated because the movement of the hand was obstructed in some way.

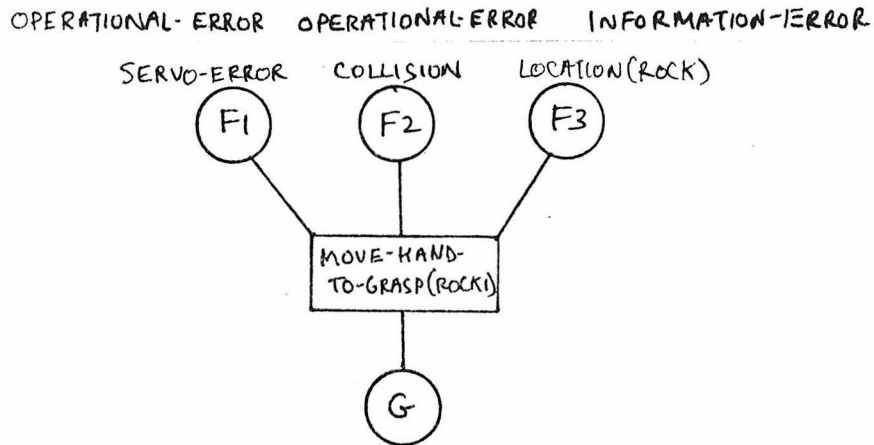


FIGURE 4.8

As before MEND starts with the possibilities represented in Figure 4.3. However, in this case verification of the preconditions and constraints eliminate NOT-GRASPABLE (F4), NOT-EMPTY-HAND (F5), and NOT-OPEN-HAND (F5) [Figure 4.8]. SATURATION indicates that collision into an obstacle has to be considered a possibility, but FRA-ROUTINE eliminates this by verifying that the hand is near the object to be grasped. F1 and F3 representing servo error and object location error respectively can both cause the hand to bump onto the rock, but their manifestations are slightly different. By looking at the position error along the sliding vector MEND can tell the difference. If such a positional error is found, it indicates that there was a servoing error. On the other hand its absence indicates that there was no servoing error and that failure must be attributed to other reasons. Let us consider the latter situation, the result being the elimination of SERVO-ERROR (F1). The possibilities are now represented by the structure shown in Figure 4.9.

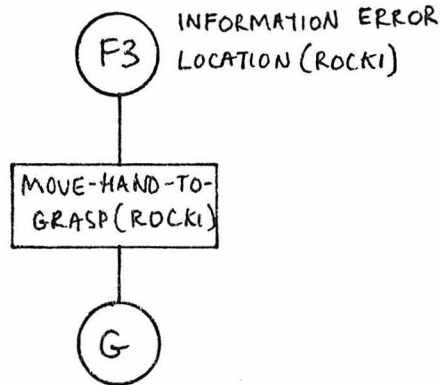


FIGURE 4.9

Since F3 represents an information error, MEND continues the analysis by looking back at the execution trace for an action which either FIND's or PROVIDE's the desired information about (LOCATION ROCK1). By simple pattern matching this action is identified as LOCATE(ROCK1). MEND looks for reasons for failure in LOCATE's failure reason model and finds:

(PTRIG (OPERATIONAL-ERROR LOCATE(ROCK) INACCURACY) NTRIG)

This fact indicates that the inaccuracy in LOCATE may go undetected. The failure tree after this part of the analysis is shown in Figure 4.10.

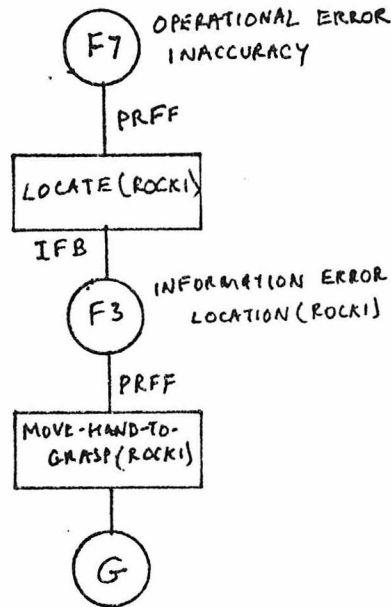


FIGURE 4.10

The PLAN-PATCHER uses the explanation represented by the failure tree in Figure 4.10 to determine the recovery plan. The UNDO-STEPS and REDO-STEPS associated with the information failure in MOVE-HAND-TO-GRASP shown below,

(UNDO-STEPS (NEEDED MHTG(ROCK) LOCATION(ROCK)) NULL-PLAN)

(REDO-STEPS (NEEDED MHTG(ROCK) LOCATION(ROCK)) MHTG(ROCK))

are used to correct the failure by patching together the plan shown in Figure 4.11.

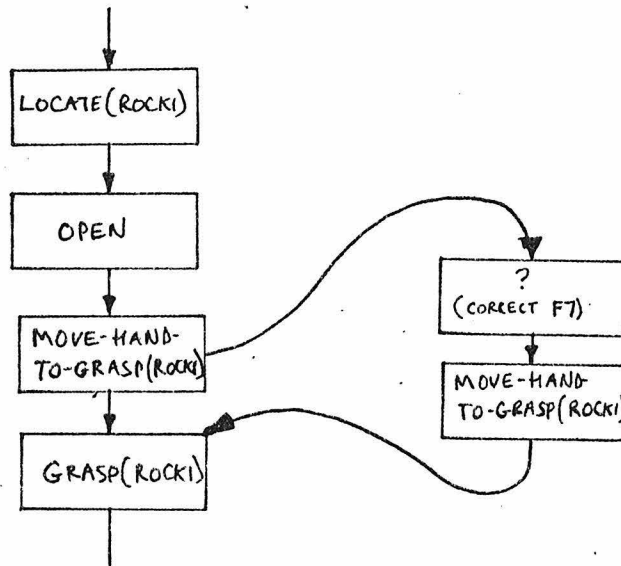


FIGURE 4.11

Continuing the recovery planning by looking at F3-LOCATE-F7 in the failure tree, the PLAN-PATCHER corrects the operational error in locate by using,

(TO-CORRECT (OPERATIONAL-ERROR LOCATE(ROCK) INACCURACY) LOC-ACC(ROCK))

where LOC-ACC : (rock)
BEGIN
APPROACH(rock);
LOCATE-RELATIVE-TO-HAND(rock)
END

to modify the structure in Figure 4.11 to that shown in Figure 4.12.

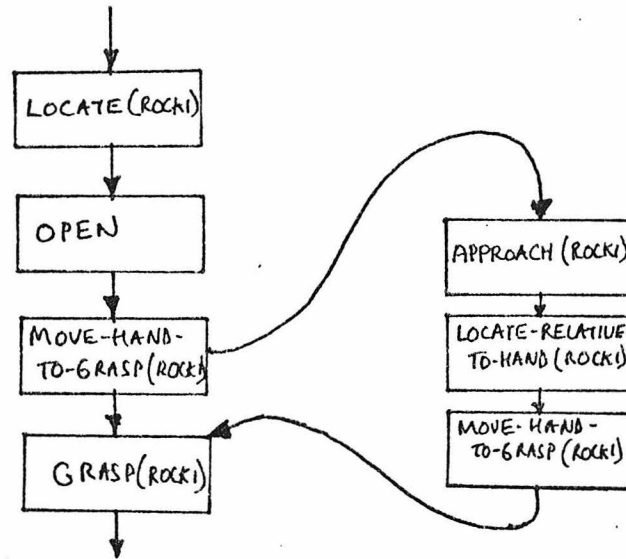


FIGURE 4.12

4.6.3 An anomalous situation

Assume that the recovery plan of Figure 4.12 is put into execution, and that the hand again bumps onto the rock. If we consider a similar situation to that in section 4.6.2, then MEND would continue the analysis represented in the tree of Figure 4.9 by producing the failure tree shown in Figure 4.13 instead of that shown in Figure 4.10. The reason for this is that the search for the action which provided the information about the location of ROCK1 will now find the recovery action LOCATE-RELATIVE-TO-HAND rather than the original LOCATE. Since there are no known failures for LOCATE-RELATIVE-TO-HAND, the chain of failure reasons G-F3-? is found to be inappropriate, and MEND has no way of explaining the failure. In such anomalous situations, MEND aborts the plan and seeks human aid in planning the recovery.

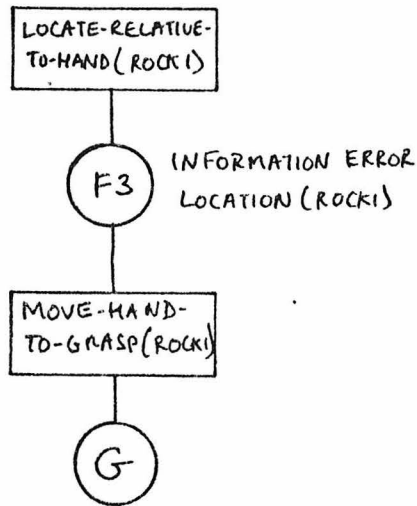


FIGURE 4.13

4.6.4 Multiple explanations

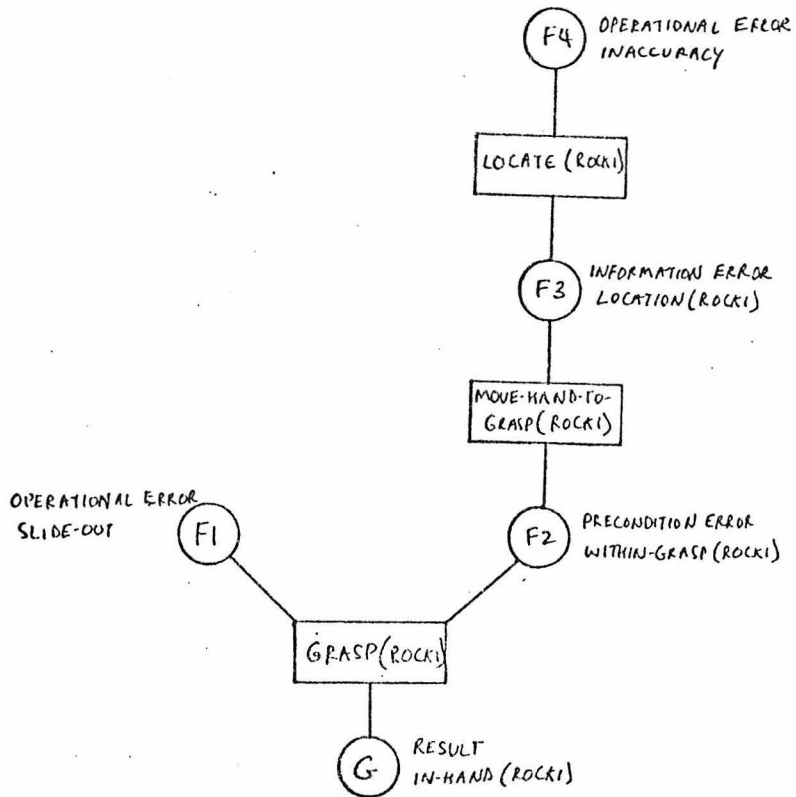


FIGURE 4.14

We consider the plan shown in Figure 4.5 again. Assume that MOVE-HAND-TO-GRASP does not trigger any failure, and that GRASP fails to find the object. The failure tree after analysis is shown in Figure 4.14.

Figure 4.14 shows two possible explanations. In such cases, the PLAN-PATCHER uses a pre-stored severity code with each failure reason to choose an explanation. The precondition error in GRASP has the greater severity code, and G-F2-F3-F4 is the preferred explanation. The corrected plan is shown in Figure 4.15.

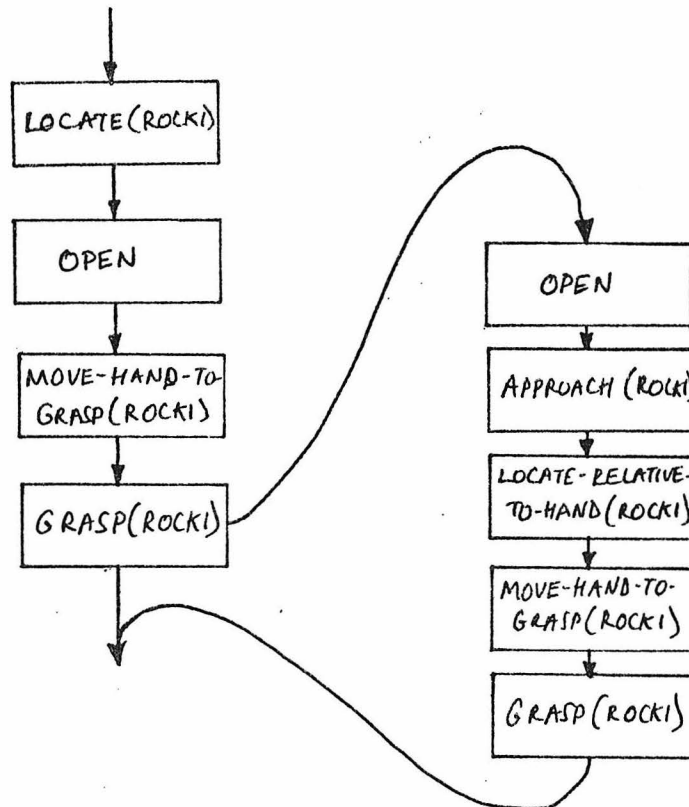


FIGURE 4.15

4.6.5 Constraint error

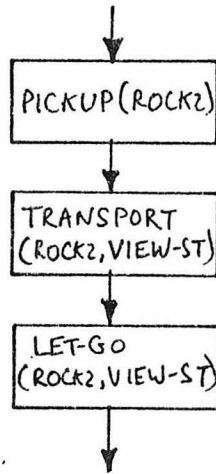


FIGURE 4.16

Figure 4.16 shows a plan which will result in ROCK2 being placed on a viewing station. Assume that the rock is not movable. TRANSPORT will trigger failure through SATURATION. The possible reasons for failure are shown in Figure 4.17.

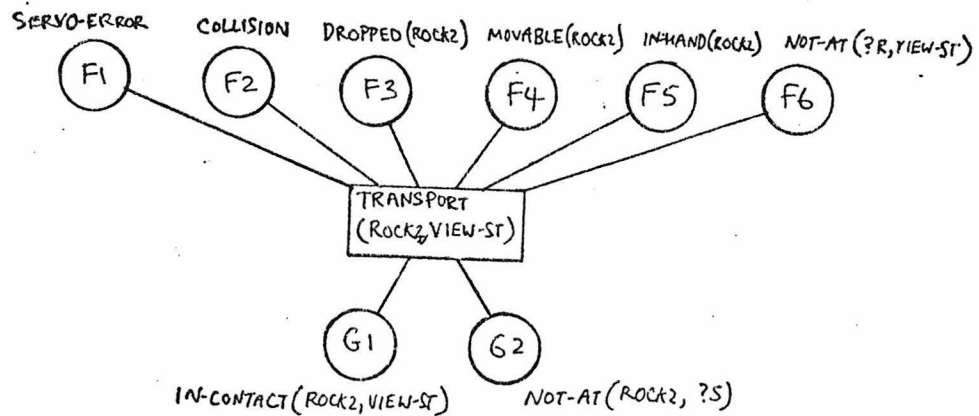


FIGURE 4.17

Assuming that preconditions have been verified, F5 and F6 can be eliminated. The TRANSPORT FRA-ROUTINE will immediately remove F3 from consideration, because it will find that the hands touch sensors are still activated. Finding that the hand has not moved, F1 and F2 are rejected as possible reasons for failure. This leaves F4, the constraint error, as the only possible explanation. The plan is aborted, since nothing can be done about constraint errors.

4.7 COMMENTS AND COMPARISONS

The main advantage of failure reason analysis as presented here is that it provides a method of directly focussing attention on the source of the failure, by using knowledge about actions, the manner in which they fail, and the history of previous actions.

Traditionally robot systems have merely dealt with the problem by trying to plan to the "nearest" intermediate subgoal, as for instance in the SHAKEY system. This method places the entire burden of recovery from failures on the planner and no advantage is taken of the the information implicit in the reason for failure. In this context of planning, we can think of the results of failure reason analysis as constraining the search space of solutions that the planner has to deal with.

Sussman has tackled the problem of failure reason analysis but in a slightly different context. The reason for (or underlying cause of) failures that he considers are domain independent goal interactions,

occurring because of incomplete knowledge about the world. MEND assumes that knowledge about the world is correctly represented in the system and further that the plan being executed is conceptually correct. Nevertheless failures occur because of operational errors of various sorts, something which is not dealt with by Sussman in the HACKER system.

Similar comments apply to Goldstein's system MYCROFT[Goldstein 74] which debugs programs in the turtle world of line drawings. An additional difference is that there is no real notion of execution monitoring. Execution proceeds to completion and failure to produce the desired line drawing is detected at the very end.

Sacerdoti[75] gives an example of NOAH's approach towards error recovery. The hierarchical approach taken in identifying the reason for failure is perhaps the best that can be done when no knowledge is available about why actions fail and how such failure manifests itself. But, I believe that it is possible to build in such knowledge into any system, to effectively aid the planner in deciding what to do.

Sacerdoti also discusses a problem involving postconditions, and its use in execution monitoring. He points out as an example that a robot should check that a pulley it has installed does not wobble. The FINISH-UP-ROUTINES in MEND are expressly intended to check postconditions. Furthermore, if the reason for the wobble is considered to be completely local to the installation process, it can be handled trivially in MEND by modelling the wobble as an operational error in the INSTALL-PULLEY action, and by providing appropriate recovery procedures.

In fact, if we consider the wobble to be an unlikely event then we can even allow the robot to continue its assembly task and wait for the wobble to manifest itself, and then correct the problem.

Certainly the major limitation in MEND is the simple minded approach taken by the PLAN-PATCHER. It has no real understanding of why its plans work and merely pieces together these actions. Furthermore, it has no concept of the the higher level goals to which the plan is addressed (as for instance, that a rock is being picked up and transported to an instrument station and measurements are going to be taken, etc.). In this respect, the Haye's system and Sacerdoti's system, NOAH, are far better, and something like those mechanisms are necessary in making MEND adaptable to more complex domains.

In tracing back precondition errors (or information errors), MEND reasons that the precondition is not true, perhaps because it was never achieved by a previous action. Restricting failure reason analysis to only this kind of reasoning has certain limitations that are discussed next.

A simple case that is not handled by MEND is one in which a precondition is achieved but later destroyed by the execution of a subsequent step. It would not be too difficult to consider extending the tree building program to look for actions which may destroy a precondition established by a previous step. However, before making such an extension, we can ask whether it is really necessary for the kind of system we have in mind. I think that the need for such a mechanism is doubtful because such interactions problems should have

been avoided in the first place when planning the task.

In a dynamic world, preconditions could become false as a consequence of the activities of other agents of change. We know very little about how to take account of the intentions of other agents who may or may not cooperate in the execution of tasks, and extensions in this direction are likely to be quite difficult.

Finally, it would be more appropriate to deal with a certain subset of precondition errors as mechanism failures. If we find that a car cannot be driven to the airport, we could attribute the failure to a precondition error (or constraint error) of the car being NOT-DRIVABLE. It does not make sense to ask whether any previous action was taken to make the car DRIVABLE. The more appropriate question is what in the mechanism of the car has failed and why it caused the failure. It would then be possible to deal with the failure of the cars by fixing, for instance, the carburetor instead of finding alternative means of transportation. Rieger[76] has developed schemes for representing the functioning of mechanisms, and these can perhaps be used for recovery from mechanism failures. Extensions to handle these cases will be useful for the JPL robot, where we could imagine situations in which the robot responds to failure by replacing the hand affixed to the manipulator by another.

MULTIPLE OUTCOME ANALYSIS

5.1 INTRODUCTION

The previous chapter discussed one approach to the error recovery problem. Failure reason analysis attempts to find an explanation for the failure. The actions to be taken to recover from the failure were derived in a relatively straightforward manner from the failure tree representing the results of the analysis.

This chapter suggests an alternative view of the problem. The historical cause of the failure is deemphasized. In contrast, attention is focussed on the actual state of the world after failure has occurred, and on how this state differs from the expected state of the world.

In order for a system like MEND to capitalize on this in formulating recovery from errors, it needs a model of what the outcomes of an action can be, how to tell them apart, and finally what can be done about it. This model will be referred to as the multiple outcome model and in the current implementation is implicit in a MOA-ROUTINE associated with each action.

The main computational steps in the process of multiple outcome analysis are shown in the Figure 5.1. We will merely illustrate multiple outcome analysis by discussing recovery from failure in MOVE-HAND-TO-GRASP and GRASP.

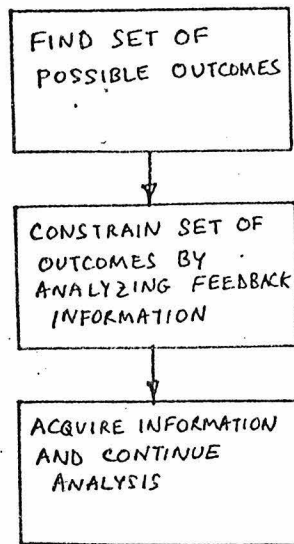


FIGURE 5.1

5.2 AN EXAMPLE : MOVE-HAND-TO-GRASP

Failure reason analysis for MOVE-HAND-TO-GRASP has already been illustrated in chapter 4. Here we will consider multiple outcome analysis for the same action. A subsequent section will compare these two schemes and suggest ways of integrating the two techniques.

5.2.1 The set of possible cases

The set of possible outcomes of an action constitutes the first aspect of the multiple outcome model. The classification of the possible states of the world resulting from execution of MOVE-HAND-TO-GRASP into qualitatively distinct outcomes is shown in Figure 5.2.

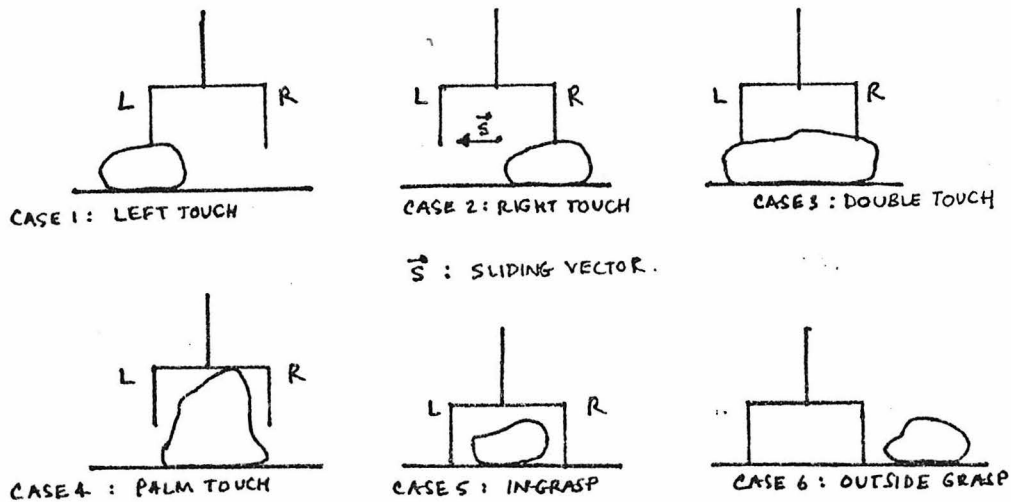


FIGURE 5.2

For reasons of clarity the graphical representation shows only two-dimensional picture of the hand, though the implementation is designed to handle the three-dimensional case. Each category represents an infinity of possible situations, but it is easy to qualitatively distinguish between the cases. LEFT-TOUCH and RIGHT-TOUCH are identical except for the fact that the finger in contact with the object is the left finger in case 1 and the right finger in case 2. We define the left finger to be that which lies in the direction of the sliding vector. DOUBLE-TOUCH occurs when both fingers contact an object which is wider than the maximum finger separation. In such a case there is no way of grasping the object in that particular orientation. PALM-TOUCH can occur only with an object whose height is greater than the length of the fingers. The IN-GRASP case is only marginally erroneous in that the hand bumps into the ground because of overshoot. Finally, OUTSIDE-GRASP is a catchall which captures all cases where the object is not within grasp. No special significance is to be attached to the fact that the

object is shown on the right of the hand in the figure for this particular case.

It is important to ask why we do not include cases such as the one shown in Figure 5.3.

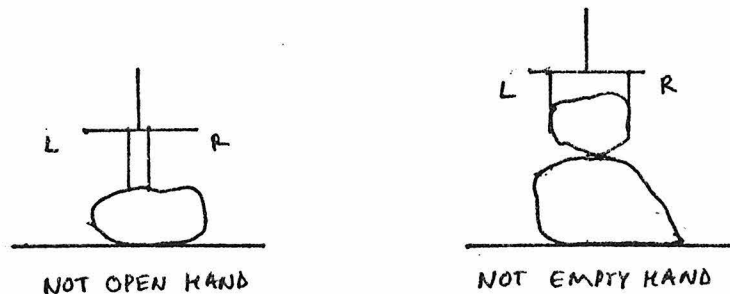


FIGURE 5.3

Clearly cases 1 - 6 are ones which can occur only when the fingers are completely open. This suffices for our purposes since precondition testing will verify that the fingers are open. If not, an action will be executed which will result in opening the fingers.

5.2.2 Feedback information and analysis of cases

There are two steps to the process of distinguishing between possible outcomes. The first is the analysis of immediately available information and the second is the acquiring of new information necessary for analysis. Both of these steps are based on knowledge about distinctive features of each outcome and this knowledge is again part of the multiple outcome model.

Feedback information that is relevant for the analysis of failure in the MOVE-HAND-TO-GRASP is given by the hand state descriptor. Among other things it specifies the coordinates of the hand and the state of the touch sensors. Combined with data about the object, this information can be used to eliminate some of the six cases from consideration. We know, for example, in case 1 that the coordinates of the hand must be approximately the same as the height of the object. If this is not the case, then clearly we need not consider this case any longer.

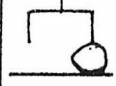
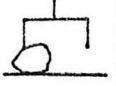
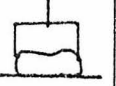
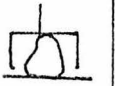
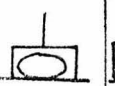
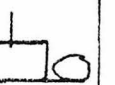
						
HAND ON OBJECT						
PALM ON OBJECT						
HAND ON GROUND						
LEFT TOUCH	-			-		
RIGHT TOUCH		-		-		

FIGURE 5.4

Figure 5.4 above summarizes the relationships between test conditions and outcomes. A "1" in the (i,j)th entry in the table indicates that test condition T(i) must be true of outcome O(j), and a "-1" indicates that T(i) must not be true. Several algorithms are possible for using the information in the table in eliminating choices from consideration. The algorithm implemented in MEND is graphically depicted in Figure 5.5.











TEST		ELIMINATE		
HAND ON OBJECT	+			
	-			
PALM-ON OBJECT	+			
	-			
HAND-ON-GROUND	+			
	-			
LEFT TOUCH	+			
	-			
RIGHT TOUCH	+			
	-			

FIGURE 5.5

This algorithm works by eliminating choices on the basis of tests performed sequentially. Certain tests will be unnecessary in that they will eliminate cases which have already been dropped from consideration.

The result of this process of analysing the feedback information is a smaller set of possible outcomes. If the number of cases has been limited to a single case then recovery planning can be initiated without any further step. It is possible though that the set of possible outcomes is larger than one. Under these circumstances MEND cannot plan the recovery. It needs to acquire more information than is immediately available through feedback, to further distinguish between the cases. This will be the subject of the next section.

5.2.3 Acquiring information and further analysis

The information available directly through feedback will in cases be inadequate for the purposes of identifying the state of the world as one of the predefined set of possible outcomes. In such cases selected actions can be executed to acquire more information.

One of the things that the robot can do to find out the state of the world after failure in MOVE-HAND-TO-GRASP is to use vision. There are both conceptual and pragmatic reasons for avoiding this. The conceptual objection is that a universal mechanism is being applied without any attempt to make maximal use of the information that is already available. A partial step towards avoiding this would be to make available to the vision system a model of what it can expect to find. In this particular case this model will be the set of possible outcomes of the MOVE-HAND-TO-GRASP action, restricted by the analysis of feedback information. However, the use of such models in simplifying the complexity of vision analysis is not a very well understood problem.

From a pragmatic point of view, the vision system that is currently part of the robot is a complex program which is slow in yielding results. Thus it makes sense to avoid use of vision if information can be acquired by other means which are simpler from a practical point of view. Figure 5.6 shows the tests appropriate for the MOVE-HAND-TO-GRASP. The tests are all simple ARM actions that yield useful information.

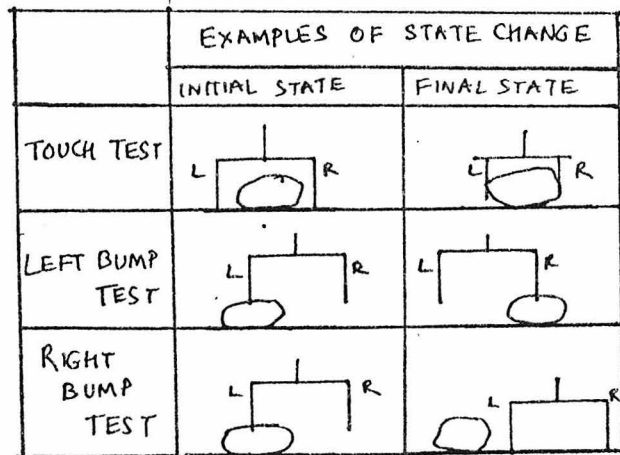


FIGURE 5.6

The TOUCH-TEST simply closes the fingers and checks whether the touch sensors are activated. The LEFT-BUMP-TEST moves the hand up by a small amount, moves "left" along the sliding vector by an amount equal to the finger separation, and then moves down. If the hand bumps onto something at a non trivial distance above the ground, then the test gives a positive result. The RIGHT-BUMP-TEST is analogous, with the hand being moved in the opposite direction.

If IN-GRASP or PALM-TOUCH [Figure 5.2] belong to the set of possibilities, then the TOUCH-TEST is applied. If the object is detected by the touch sensors, MEND recognizes that the problem has been solved, and no more tests are performed. If the touch sensors are not activated, then the above two cases IN-GRASP and PALM-TOUCH are eliminated from the set of possibilities. Regardless of the result of the test, the fingers are opened. The other tests can give unpredictable results if OUTSIDE-GRASP is among the set of possibilities. In such circumstances, MEND resorts to vision. If OUTSIDE-GRASP has been eliminated by analysis of feedback information and LEFT-TOUCH belongs to the set of possible outcomes, then the LEFT-BUMP-TEST is run. If the test gives a positive result, then case 2 (RIGHT-TOUCH) is eliminated. Otherwise, case 1 (LEFT-TOUCH) case is removed from consideration. If any further analysis is required, then the RIGHT-BUMP-TEST is run.

5.2.4 Recovery planning

Dynamic information gathering introduces a difficulty in that the actions taken to acquire information change the state of the world. Thus one of the criteria to be placed on the information gathering actions is that their effects be simple to characterize. Further, these actions should have a high probability of success and should not cause any failures of their own.

The state changes that result do not necessarily make the recovery steps more complex. In fact, there are cases when the information gathering steps actually produce desirable side effects. For example if the result of the TOUCH-TEST is positive, MEND has not only identified the error state but has further achieved a state from which success in grasping the object can be almost guaranteed.

Figure 5.7 shows the recovery actions to be taken on identification of the error state. They need no detailed explanation, and the only point to note is the following: The parameter for the search routines specifies the starting point of the search. By noting the hand state when the failure occurred and using it as the starting point of the search, the search strategies can be made independent of any information gathering steps that may have been executed.

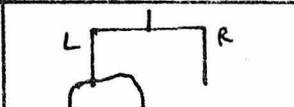
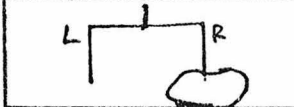
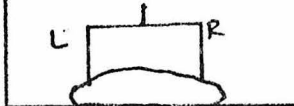
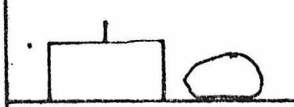
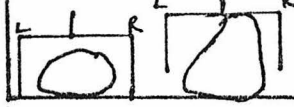
CASE	RECOVERY STEP
	SEARCH LEFT
	SEARCH RIGHT
	ATTEMPT GRASP ALONG ↓ DIRECTION
	USE VISION
	—

FIGURE 5.7

5.2.5 Anamolous Situations

In our discussion in the previous sections, we have ignored the possibility of several kinds of anamolous situations. The first kind of anamolous situation that can occur is the elimination of all possible cases. This can happen because of two distinct reasons. The model of possible outcomes that MEND has been provided with may not be comprehensive in categorizing all situations, or the information on which the analysis is based may be erroneous. Whatever the reason, finding a null set of possibilities, MEND will abort the recovery planning.

The possibility of keeping a record of the tests performed and their effects in eliminating cases has been considered. Such a record would allow MEND to deal with anamolous situations by reconsidering certain outcomes that have been eliminated from consideration.

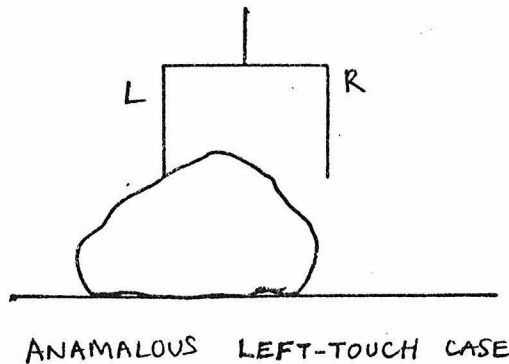


FIGURE 5.8

A second kind of anamolous situation arises when MEND's analysis points to a catalogued case which is somewhat different from the actual situation. For instance, MEND will respond to the situation illustrated

in Figure 5.8 by a SEARCH-LEFT recovery strategy, imagining it to be a simple LEFT-TOUCH situation. However, the recovery plan will fail because the object is too large and hence not graspable.

5.3 A SECOND EXAMPLE : GRASP

In general, the set of possible failure outcomes of GRASP are not very well defined. However, in cases where the failure is the result of the operational error SLIDE-OUT, the failure outcomes to be considered are shown in Figure 5.9.

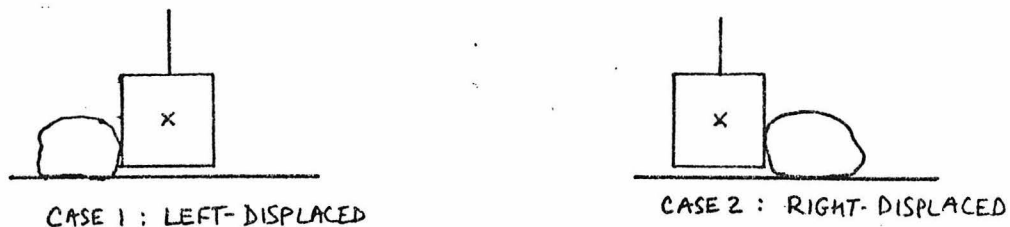


FIGURE 5.9

(In contrast to the previous figures, we show a "side" view of the hand in which we see the outside face of the fingers. The small cross in the middle of the fingers indicates that the sliding vector is pointing away from the reader.) The rationale behind the outcomes shown in figure 5.9 is graphically illustrated in Figure 5.10.

Feedback data does not provide any useful information in distinguishing between the two cases. Thus a dynamic information gathering step PISV-BUMP-TEST is resorted to. This test and its effects

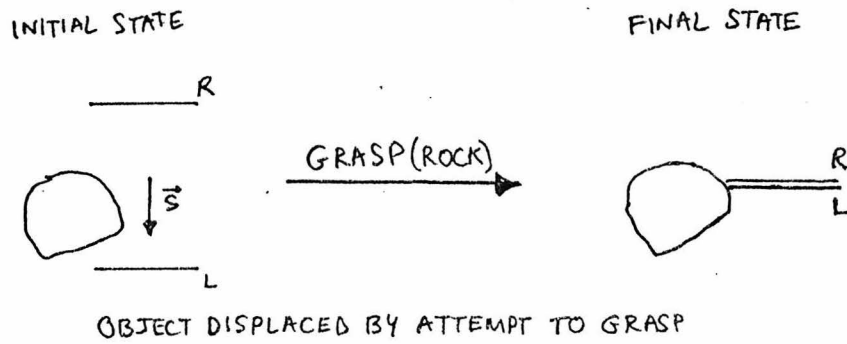


FIGURE 5.10

are illustrated in Figure 5.11. The fingers are not opened before this test is executed and this has the advantage of making the outcomes of this test simple and well defined.

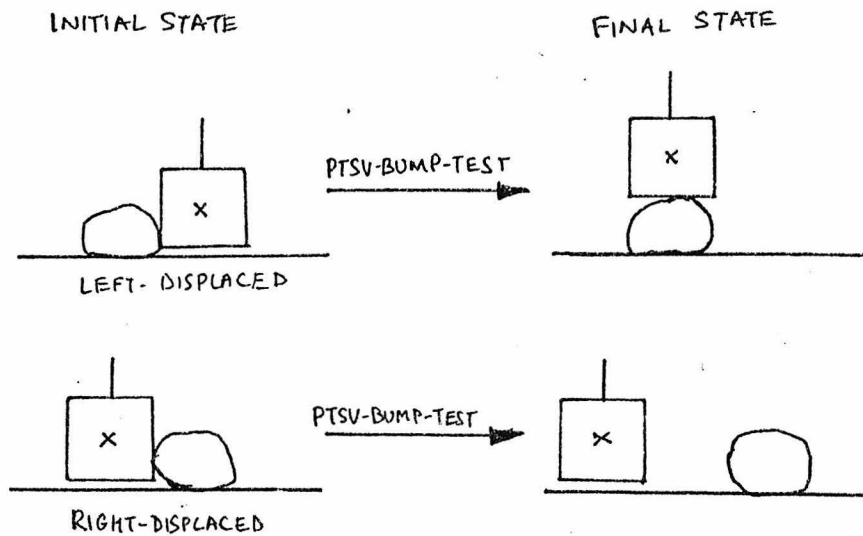


FIGURE 5.11

The PTSV-BUMP-TEST indicates whether or not the object is to the left or right of the hand viewed with respect to the sliding vector. The corrective steps are to open the hand and then to move the hand to grasp the rock at its newly estimated location.

5.4 FAILURE REASON ANALYSIS VERSUS MULTIPLE OUTCOME ANALYSIS

MEND uses both failure reason analysis and multiple outcome analysis in tackling the error recovery problem. These schemes are not to be thought of as competing strategies. Rather they complement each other in natural way, suggesting a possible integration of the two schemes. Let us first make a comparison of the two schemes.

To make our comparison concrete we again consider the performance of failure reason analysis and multiple outcome analysis for MOVE-HAND-TO-GRASP. Consider a case in which the hand bumps into the object to be grasped and FRA (the failure reason analyzer in MEND) concludes that the failure is the result of a location error. The recovery based on this analysis locates the object more accurately and attempts to reposition the hand around the object. This is quite satisfactory in the sense that it solves the problem. It is unsatisfactory, however, for two reasons. Firstly, it does not make maximal use of the available feedback information. Secondly, it does not recognize that the important thing for recovery is the relative position of the object with respect to the hand, regardless of the reason for the failure.

Multiple outcome analysis exploits the availability of feedback information, and goes even further by acquiring necessary but missing information. With such analysis it is possible to characterize the relation between the hand and the object even without accurate information about the location of the object. Such a characterization would be pointless if the robot had to eventually determine the location

of the object (using vision) in order to effect recovery. This is not the case, as the earlier example has shown, and MEND can suggest the use of specialized actions which are simple and which achieve the immediate goal of positioning the hand around the object.

The situation is not as one sided as the above argument may indicate. A characterization of the current state of the world on the basis of a detailed analysis may be unnecessarily expensive. For instance, an operational error of servoing can be corrected by simple repositioning of the hand. By ignoring the reason for failure MEND engages in a needless analysis which may involve dynamic information gathering. Multiple outcome analysis is therefore deficient in not making use of the history of the process that resulted in the current state.

5.5 INTEGRATION OF THE TWO SCHEMES

The above argument clearly indicates the need for integration of the two schemes. This has been done in MEND in a rather simple and ad hoc manner. MEND always begins by performing failure reason analysis. The results are sent over to DECIDER (Figure 3.2), which has a rather arbitrary classification of failure reasons into two categories of SIMPLE and COMPLEX. For the cases in which the failure reason has a SIMPLE recovery strategy, multiple outcome analysis is not performed. In other cases, MEND resorts to multiple outcome analysis, hoping to find a simpler recovery strategy. For the example we have been considering, SERVO-ERROR is considered a SIMPLE error and MEND recovers

by repositioning the hand. Multiple outcome analysis is not done. On the other hand location errors are tackled by multiple outcome analysis.

We have a rather ad hoc scheme based on our knowledge of what is SIMPLE and what is not. It seems unlikely that there are any general criteria for evaluating the effectiveness of these schemes so that this criteria can be used as the basis for integration. This is because it is the specific properties of the task domain which determine the effectiveness of one strategy versus another.

SUMMARY AND CONCLUSIONS

6.1 INTRODUCTION

This dissertation has addressed itself to a specific problem faced by a robot system -- namely, that of recovering from failures in execution of a plan of actions. Failures occur partly because the robot operates in a domain which cannot be characterized exactly and partly because actions the robot takes do not always function as expected.

Execution of plans needs to be monitored in order to detect errors. A very conservative execution monitoring strategy would check a large number of conditions in an exhaustive manner to avoid failures at all costs. Such an approach tends to make robot systems impractical. This suggests that execution monitoring should be restricted to simple checks, and that mechanisms should be provided for dealing with failures as and when they occur.

Traditionally, robot systems with automated planners deal with failures that have been detected by merely replanning to achieve the desired goal. A somewhat different approach has been taken in this study, and this approach provides some simple and effective techniques for dealing with the problem of error recovery.

6.2 HISTORICAL NOTE

The subject of error recovery was investigated in connection with the JPL robotics research program. Manipulative tasks proved to be a rich enough domain to illustrate several interesting aspects of this problem. The problem was tackled by designing a module called MEND, to be integrated with the existing robot software. A first version of MEND embodied a simpler scheme for error recovery than the one described in this report. This first implementation directed attention to certain limitations of the system, spurring a more detailed study of the problem. The results of this detailed study have been subsequently incorporated in the design of a second version of MEND, which is not yet an integral part of the JPL robot system.

6.3 A DISCUSSION OF THE APPROACH

The approach taken in MEND is quite different from the traditional manner in which the problem of error recovery has been tackled. Let us take a look at the differences.

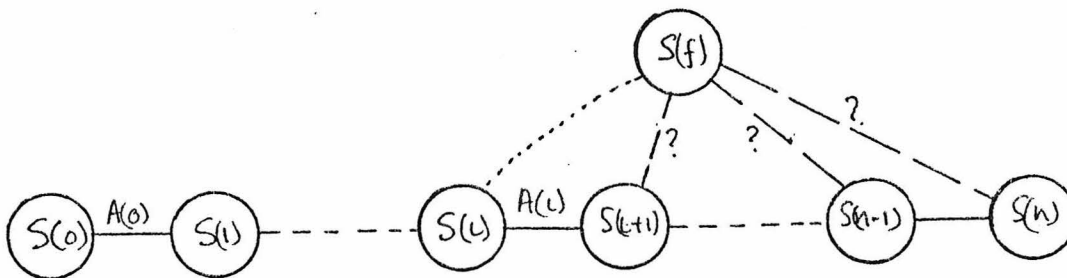


FIGURE 6.1

We can imagine the execution of a task as a sequence of transformations which changes the world from an initial state, $S(0)$, into a final state, $S(n)$, which represents the goal of the robot. Figure 6.1 depicts the intermediate states of the transformations from the $S(0)$ to $S(n)$ which result from the execution of a plan of actions, $A(0) \dots A(n-1)$. Assume that a failure is detected on execution of the action $A(i)$. This means that the robot recognizes that the action $A(i)$ has not produced the expected state $S(i+1)$, but rather has resulted in some failure state $S(f)$.

The problem of error recovery is that of going from the failure state $S(f)$ to the goal state $S(n)$. Typically other robot systems treat $S(f)$ as though it were some arbitrary state and respond to the failure by replanning to achieve the given goal. In such an approach there is no essential difference between error recovery [the transformation from $S(f)$ to $S(n)$] and planning [for instance, the transformations from $S(0)$ to $S(n)$]. Parts of the old plan can sometimes be reused, as for instance, by replanning from $S(f)$ to $S(j)$ so that the original plan can be used to go from $S(j)$ to $S(n)$.

The central idea in failure reason analysis is that finding an explanation of the failure, i.e., understanding why action $A(i)$ resulted in the state $S(f)$, can focus attention on where the problem lies and what can be done about it. With this viewpoint error recovery is seen

as something quite different from traditional planning in that we do not ask:

What can be done to go from $S(f)$ to $S(n)$?

but rather ask:

Why did action $A(i)$ result in the state $S(f)$?

Of course, even if we do understand why the failure occurred, we are still confronted with the problem of going from the state $S(f)$ to $S(n)$. MEND shows that even very simple strategies are effective in solving this latter problem, once an explanation for the failure has been found.

Where does multiple outcome analysis fit in all this? Again, as in failure reason analysis, $S(f)$ is recognized as being something special and not an arbitrary state of the world. Since we know that $S(f)$ resulted from the execution of an action, with a good model of the behavior of the action, we can determine the state $S(f)$. Specifically, the action model predicts that $S(f)$ is one of $S(f_1) \dots S(f_m)$, and the problem in multiple outcome analysis is that of characterizing the state $S(f)$. This is done by looking for distinguishing features that characterize the failure states. The planning problem has been made trivial by including recommendations of corrective steps as part of the multiple outcome model of actions.

6.4 FEATURES OF THE SOLUTION

MEND's performance as part of the robot system is based on the fact that it monitors the execution of plans and detects failures. We discuss this and then outline the two strategies used by MEND in recovering from failures.

Any system which attempts to recover from errors should have the capability of detecting errors. A robot should thus be able to judge the progress of activities in leading towards a pre-specified goal. Such monitoring of activities, however, should not be so prohibitively expensive that the robot is paralyzed by indecision.

MEND implements a simple execution monitoring strategy. Precondition testing in the INITIATE-ROUTINES incorporates knowledge about what can be tested easily and what cannot. MEND allows execution to proceed even when some preconditions have not been absolutely verified. However, these are noted down in the execution trace as being unverified so that the information can be later used in analyzing failure.

Errors are detected at two distinct levels of the system. At the hardware level, completion criteria provide the simplest test of success. Given success of the action at this level, software checks of certain simple conditions allow the detection of another class of errors. The FINISH-UP-ROUTINES are selective in applying only those tests which can be guaranteed to give results inexpensively. MEND again sacrifices completeness for efficiency, and allows certain errors to go

undetected. These errors are often detected in a simple way at subsequent steps in the execution of the plan. Thus MEND's performance does not suffer significantly in spite of its simple approach in detecting errors.

6.4.2 Failure reason analysis

The history of actions taken by the robot, the current state of the world, and the nature of the failure are all pieces of information that are useful in determining the cause of the failure. A classification of failure reasons into four basic types -- operational errors, information errors, precondition errors, and constraint errors -- proves useful in dealing with failures intelligently.

MEND analyzes the failure by building a failure tree to represent the possible explanations for failure. Several constraints are used to limit the set of all possible explanations. By keeping a record of unverified preconditions, MEND can eliminate verified preconditions as possible reasons for the failure. Knowledge about the manifestation of the failure is used to further limit the set of possibilities. Finally, some explanations are shown to be impossible on the basis of the history of actions.

Having found an explanation of the failure, MEND patches a recovery plan based on very simple strategies appropriate to the different kinds of failures. Even such simple strategies prove adequate in producing recovery plans which are contextually appropriate.

6.4.3 Multiple outcome analysis

MEND's capabilities are significantly enhanced by a second scheme of error recovery. This is based on a detailed model of the possible states of the world when an error has occurred. The states resulting from the execution of an action are characterized into a small number of qualitatively distinct outcomes.

MEND analyzes feedback information in limiting the set of possible cases. A sequence of prefabricated tests eliminate cases from consideration. Often such analysis will be adequate to identify the outcomes of an action, in which case MEND has all the information it needs to plan recovery.

There are cases, however, when such analysis is inadequate to trim the choices down to a single case. Under these circumstances MEND executes actions as a means of gathering specific information that is useful in further disambiguating the choices.

The result of the analysis is the identification of the failure outcome. Each failure outcome is associated with an appropriate recovery strategy, so that recovery from the failure is simple and immediate.

6.5 CONTRIBUTIONS

The techniques of planning recovery from failures through failure reason analysis and multiple outcome analysis are contributions to the subject of robotics. More importantly, however, the problem of error recovery is recognized to be a member of a larger class of problems involving knowledge representation and common sense reasoning, both of which are core topics in the study of artificial intelligence. The solution presented in this thesis makes some new contributions to these core topics.

In regard to knowledge representations, this study has established certain guidelines for the structuring of knowledge about actions. Traditionally, actions have been simply modelled in terms of preconditions and postconditions. This study has shown that by extending the model of actions to include a failure reason model and a multiple outcome model, a robot can more directly address itself to the problem of error recovery.

The failure reason model is a means of representing the knowledge about why actions fail, knowledge about their manifestation, and finally knowledge about what can be done to correct failures. The classification of failure reasons is useful in structuring the knowledge about actions into categories that can each be easily and separately dealt with.

The multiple outcome model represents the possible outcomes of an action, knowledge about how they can be distinguished, and about what can be done about them. Such a model provides a systematic way of exploiting feedback information, that is typically available in any robot system.

Failure reason analysis and multiple outcome analysis can be viewed in rather general terms as specialized techniques for performing common sense reasoning appropriate for planning recovery from failures. The above models form an essential part of the knowledge base necessary for such reasoning.

6.6 FUTURE WORK

The results reported in this thesis provide a solution to the problem of error recovery in robot systems that are relatively simple in some respects. Even for more complex robot systems, the techniques of failure reason analysis and multiple outcome analysis are likely to be very useful, but the limitations of MEND suggest directions for future efforts.

The recovery strategies investigated have been limited to those which achieve the intended effects of the failed action. More generally, an intelligent robot will have to consider trying other alternatives which may equally well achieve the goal. For instance, finding that a rock cannot be moved, a robot interested in sample collection may look for another rock with similar properties. Such

strategies require a representation of the larger set of goals to which a specific plan is addressed. Failure reason analysis is likely to play an important role in planning recovery, even in robot systems with more complex goal structures than the ones considered in this study.

In general, recovery planning becomes expensive if the robot abandons the whole plan because of a failure in one step. By keeping interrelationships between different steps in the plan in an explicit form, it is possible to restructure and build upon the old plan in planning recovery. Some preliminary ideas in this direction have been investigated but they need to be implemented as part of the system before it becomes clear how well such a scheme will work.

REFERENCES

- [Bejczy] A.Bejczy, WORK SPACE ANALYSIS FOR VEHICLES MOUNTED ARM, California Institute of Technology, Jet Propulsion Laboratory, G&C Technical Memo 165, June 1972.
- [Choate 72] R.Choate and L.D.Jaffe, SCIENCE ASPECTS OF A REMOTELY CONTROLLED MARS SURFACE ROVING VEHICLE, California Institute of Technology, JPL Technical Report 760-76, July 1972.
- [Dobrotin 73] B.M.Dobrotin and V.D.Scheinman, DESIGN OF COMPUTER CONTROLLED MANIPULATOR FOR ROBOT RESEARCH, Third International Joint Conference on Artificial Intelligence, Stanford, August 1973.
- [Feldman 71a] J.Feldman, and R.F.Sproull, SYSTEM SUPPORT FOR THE STANFORD HAND-EYE SYSTEM, Second International Joint Conference on Artificial Intelligence, London, September 1971.
- [Feldman 71b] J.Feldman, et.al., THE USE OF VISION AND MANIPULATION TO SOLVE THE 'INSTANT INSANITY' PUZZLE, Second International Joint Conference on Artificial Intelligence, London, September 71.
- [Fikes 71] R.E.Fikes, MONITORED EXECUTION OF ROBOT PLANS PRODUCED BY STRIPS, SRI Artificial Intelligence center, Technical Note 55, April 1971.

[Finkel 74] R.Finkel, et.al., AL, A PROGRAMMING SYSTEM FOR AUTOMATION,
SRI Artificial Intelligence Center, Final Report, October 1971.

[Goldstein 74] I.P.Goldstein, UNDERSTANDING SIMPLE PICTURE PROGRAMS, MIT
Artificial Intelligence Laboratories, Technical Report 294,
September 74.

[Hayes 75] P.J.Hayes, A REPRESENTATION OF ROBOT PLANS, Fourth
International Joint Conference on Artificial Intelligence,
Technical Note 76, April 1973.

[Hooke 74] A.A.Hooke, B.T.Larman, W.M.Whitney, THE IMPACT OF ROBOTS ON
PLANETARY MISSION OPERATIONS, International Telemetering
Conference, Los Angeles, October 74.

[Lewis 74] R.A.Lewis, AUTONOMOUS MANIPULATION OF A ROBOT: SUMMARY OF
MANIPULATOR SOFTWARE FUNCTIONS, California Institute of Technology,
JPL Technical Report 33-679, March 74.

[Lewis] R.A.Lewis, PRACTICAL MANIPULATOR SOFTWARE, Proceedings of the
IEEE, Symposium on Automatic Computation and Control, Milwaukee,
April 1976.

[Martin 74] W.A.Martin, A SYSTEM FOR BUILDING EXPERT PROBLEM SOLVING
SYSTEMS INVOLVING VERBAL REASONING, OWL Notes, MIT, 1974.

[Nilsson 73] N.J.Nilsson, A HIERARCHICAL ROBOT PLANNING AND EXECUTION SYSTEM, SRI Artificial Intelligence Center, Technical Note 76, April 73.

[Raphael 71] B.Raphael, et.al., RESEARCH AND APPLICATIONS - ARTIFICIAL INTELLIGENCE, SRI Artificial Intelligence Center, Final Report, October 71.

[Rieger 75] C.Rieger, ONE SYSTEM FOR TWO TASKS : A COMMONSENSE ALGORITHM MEMORY THAT SOLVES PROBLEMS AND COMPREHENDS LANGUAGE, Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR, September 1975.

[Roth 75] S.D.Roth and K.A.Stevens, USER MANUAL FOR THE PDP-10 ROBOT SOFTWARE, California Institute of Technology, Information Science, A.I.Note, February 77.

[Sacerdoti 75] E.D.Sacerdoti, A STRUCTURE FOR PLANS AND BEHAVIOR, SRI Artificial Intelligence Center, Technical Note 109, August 75.

[Srinivas 73a] S.Srinivas, DESIGN OF A ROBOT EXECUTIVE SYSTEM, California Institute of Technology, Information Science, A.I.Note 2, October 73.

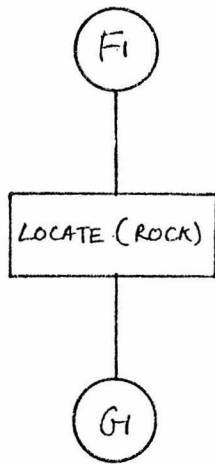
- [Srinivas 73b] S.Srinivas, ROBOT EXECUTIVE SYSTEM, California Institute of Technology, Information Science, A.I.Note 3, November 1973.
- [Stevens 74] K.A.Stevens, ROBOT IMPLEMENTATION MANUAL, California Institute of Technology, Information Science, A.I.Note 26, May 74.
- [Sussman 73] G.J.Sussman, A COMPUTATIONAL MODEL OF SKILL ACQUISITION, MIT Artificial Intelligence laboratory, Technical Report 297, August 73.
- [Udupa 74] S.M.Udupa, THE DESIGN OF A ROBOT NAVIGATOR, California Institute of Technology, Information Science, A.I.Note 20, 21, March 1974.
- [Udupa 76] S.M.Udupa, COLLISION DETECTION AND AVOIDANCE IN COMPUTER CONTROLLED MANIPULATORS, California Institute of Technology, Information Science, Doctoral Dissertation, September 74.
- [VanLehn 73] K.A.VanLehn, SAIL USER MANUAL, Stanford Artificial Intelligence Laboratory, Memo 204, July 1973.
- [Weinstein 75a] M.Weinstein, A FRAMEWORK FOR ROBOTIC DESIGN, California Institute of Technology, Information Science, Technical Report 14, June 1975.

[Weinstein 75b] M.Weinstein, STRUCTURED ROBOTICS, California Institute of Technology, Information Science, Technical Report 15, June 1975.

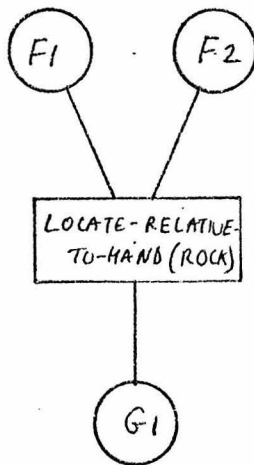
[Whitney 74] W.M.Whitney, HUMAN VERSUS AUTONOMOUS CONTROL OF PLANETARY ROVING VEHICLES, IEEE Symposium on System, Man and Cybernetics, Dallas, October 74.

APPENDIX 1

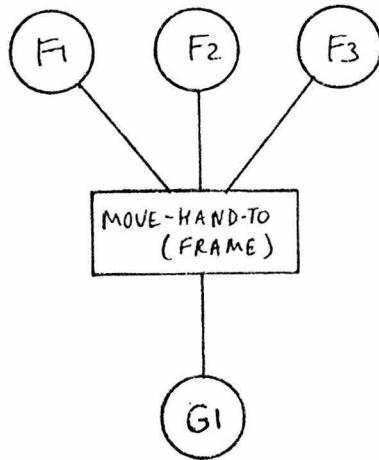
ACTION MODELS IN GRAPHIC FORM



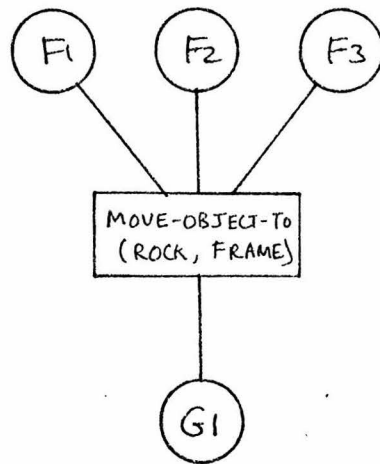
F1 = OE : INACCURACY
G1 = FINDS : LOCATION (ROCK)



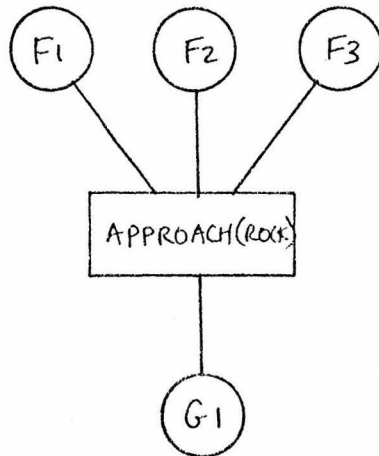
F1 = PE : HAND-AT-APPROACH-POINT (ROCK)
F2 = IE : LOCATION (ROCK)
G1 = FINDS : LOCATION (ROCK)



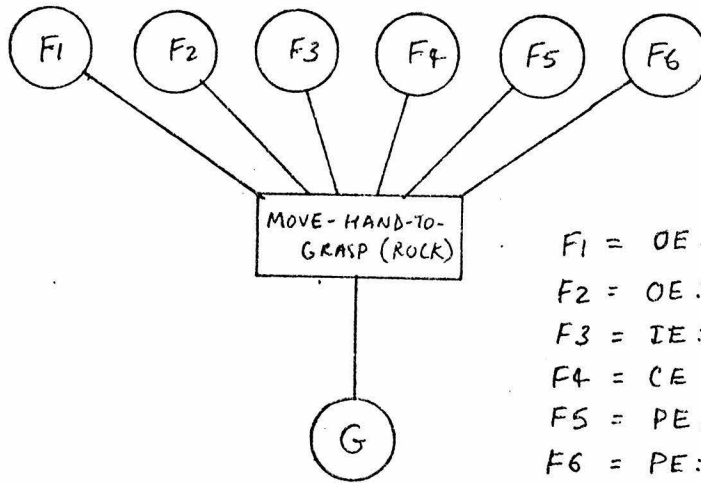
F3 = PE : EMPTY-HAND
F1 = OE : SERVO-ERROR
F2 = OE : COLLISION
G1 = RESULT : HAND-AT (FRAME)



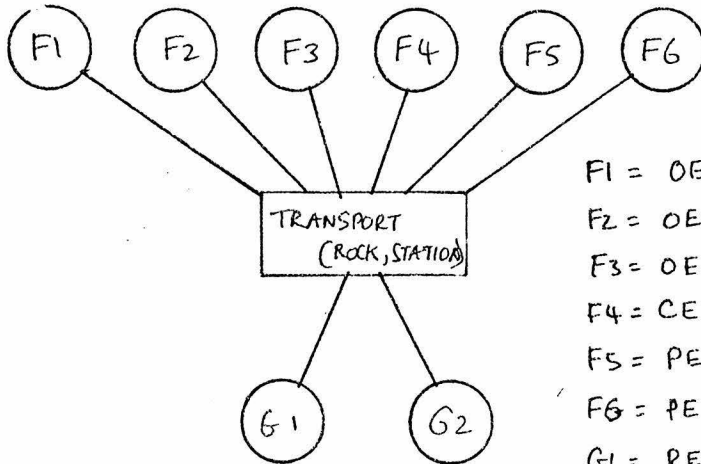
F3 = PE : IN-HAND (ROCK)
F1 = OE : SERVO-ERROR
F2 = OE : COLLISION
G1 = RESULT : HAND-AT (FRAME)



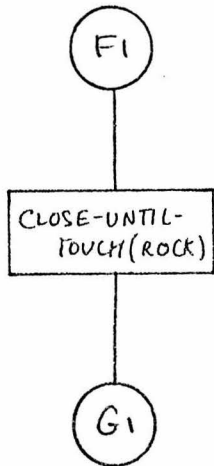
F1 = OE : SERVO-ERROR
F2 = OE : COLLISION
F3 = IE : LOCATION (ROCK)
G1 = RESULT : HAND-AT-APPROACH POINT (ROCK)



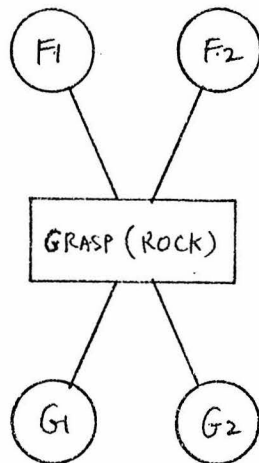
- F1 = OE: SERVO-ERROR
- F2 = OE: COLLISION
- F3 = IE: LOCATION(ROCK)
- F4 = CE: GRASPABLE(ROCK)
- F5 = PE: EMPTY-HAND
- F6 = PE: OPEN-HAND
- G = RESULT: WITHIN-GRASP(ROCK)



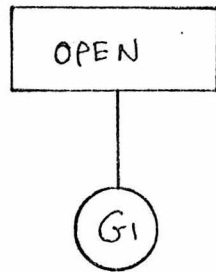
- F1 = OE: SERVO-ERROR
- F2 = OE: COLLISION
- F3 = OE: PROPPED(ROCK)
- F4 = CE: MOVABLE(ROCK)
- F5 = PE: IN-HAND(ROCK)
- F6 = PE: NOT-AT(?R, STATION)
- G1 = RESULT: IN-CONTACT(ROCK, STATION)
- G2 = RESULT: NOT-AT(ROCK, ?S)



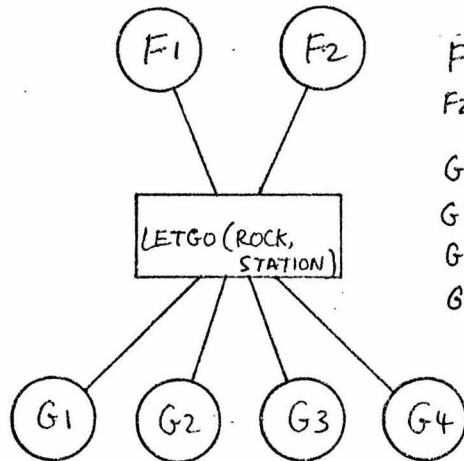
$F_1 = PE : WITHIN-GRASP(ROCK)$
 $G_1 = RESULT : TOUCHING-HAND(ROCK)$



$F_1 = OE : SLIDE-OUT$
 $F_2 = PE : WITHIN-GRASP(ROCK)$
 $G_1 = RESULT : IN-HAND(ROCK)$
 $G_2 = RESULT : TOUCHING-HAND(ROCK)$



G1 : RESULT : OPEN-HAND



F1 = PC : IN-CONTACT (ROCK, STATION)

F2 = PC : TOUCHING-HAND (ROCK)

G1 = RESULT : AT (ROCK, STATION)

G2 = RESULT : EMPTY-HAND

G3 = RESULT : OPEN-HAND

G4 = PROVIDES : LOCATION (ROCK)

APPENDIX 2

A COMBINATORIAL PROBLEM

In multiple outcome analysis we are interested in identifying the failure state as one of a preconceived set of possible outcomes of an action. Identification involves the testing of conditions that allow one outcome to be distinguished from another. From a theoretical standpoint one could ask for the minimum number of tests that need to be performed. This problem can be formulated in combinatorial terms, and has applications in wide variety of situations.

Let S be a set of n states, S₁, S₂, ... , S_n, and let T be a set of m tests, T₁, T₂, ... , T_m. Construct a matrix A of m rows and n columns with entries (-1, 0, 1) in the following manner:

A(i, j) = 1 if test T_i = TRUE in state S_j

A(i, j) = -1 if test T_i = FALSE in state S_j

A(i, j) = 0 otherwise.

We define such a matrix to be sufficient if:

$$\forall j \substack{1 \leq j \leq n} \quad \forall k \substack{1 \leq k \leq n} \quad \exists i \substack{1 \leq i \leq m} \quad ((A(i, j) = 1) \text{ and } A(i, k) = -1) \text{ or} \\ k \neq j \quad (A(i, j) = -1 \text{ and } A(i, k) = 1)$$

The intuitive interpretation of the above definition is that the set of

tests is sufficient, if for any pair of states there exists at least one test which gives different results, thus allowing the states to be distinguished one from the other.

Given a matrix A , the problem under consideration is that of finding a submatrix B (formed by deleting rows of A) with m' rows and n columns, which is sufficient and has a minimal number of rows. Many interesting questions about the properties of such a minimal matrix arise, but we are primarily interested in an efficient computational procedure that will find such a minimal submatrix.