

**Interweaving Reason,
Action and Perception**

Claude L. Fennema, Jr.

COINS TR91-56

September 1991

This research was supported by the Defense Advanced Research Projects Agency under contract numbers DAAE07-91-C-RO35 and DACA76-85-C-0008, and by the National Science Foundation under grant number CDA-8922572.

INTERWEAVING REASON, ACTION AND PERCEPTION

A Dissertation Presented

by

CLAUDE L. FENNEMA, JR.

**Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY**

September 1991

Department of Computer and Information Science

© Copyright by Claude L. Fennema, Jr. 1991

All Rights Reserved

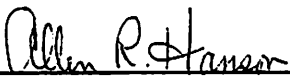
INTERWEAVING REASON, ACTION AND PERCEPTION

A Dissertation Presented

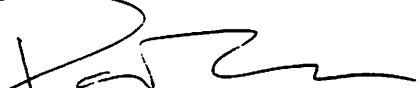
by

CLAUDE L. FENNEMA, JR.

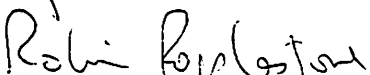
Approved as to style and content by:



Allen R. Hanson, Chair




Paul R. Cohen, Member



Robin J. Popplestone, Member



Theodore E. Djaferis, Member



W. Richards Adrion, Department Chair,
Department of Computer and Information Science.

To Betsy, Kate and Christy

ABSTRACT

INTERWEAVING REASON, ACTION AND PERCEPTION

September 1991

Claude L. Fennema, Jr.,

B.S., Massachusetts Institute of Technology

M.A., Johns Hopkins University

Ph.D., University of Massachusetts

Directed by: Professor Allen R. Hanson

In their attempt to understand and emulate intelligent behavior Artificial Intelligence researchers have taken a reductionist approach and divided their investigation into separate studies of reason, perception and action. As a consequence, intelligent robots have been constructed using a coarse grained architecture; reasoning, perception and action have been implemented as separate modules that interact infrequently. This dissertation describes an investigation into the effect of reducing this architecture granularity on the efficiency of the overall system. The thesis takes the position that significant computational efficiencies can be gained by introducing a fine grained integration or "interweaving" of these functions and demonstrates these savings for an intelligent navigation system.

The dissertation develops a paradigm referred to as the "reason a little, move a little, look a little", or RML paradigm, and describes a specific RML implementation used for experimentation. The results from this work show significant complexity reduction for planning and vision due to the RML paradigm. It is argued that the replanning triggered by the realities of carrying out actions in a real environment cause the planner in a coarse grained system to exhibit a complexity of $O(\rho^2)$ (ρ = path length) whereas the fine grained RML system exhibits a complexity between $O(\rho \log(\rho))$ and $O(\rho)$. It is then argued that the complexity of vision in the RML system is $O\left[\max\left[n*m, \frac{n}{s^2}\right]\right]$ where $n*m$ is a measure of the complexity of the *local* environment and s is the number of vision invocations per foot. Finally, it is shown that the RML system results in accurate action execution.

In addition to this global result, the design details of the experimental system reveal novel approaches to representation, planning and vision. The environment is represented as a network which organizes space into a "contained-by" hierarchy which is used as a mechanism to cope with positional uncertainty and to focus planning computations. Planning is done in three dimensions and plans are developed using a hierarchical decomposition induced by the geometry of the environment. The approach to vision is analogous to the way a blind man uses his cane: to verify that reason is *consistent* with reality.

TABLE OF CONTENTS

ABSTRACT.....	v
LIST OF FIGURES.....	x
Chapter	
1. INTRODUCTION 1	
1.1 The Problem - Navigation	2
1.2 The Approach.....	6
1.3 Applications	9
1.3.1 The need for an application oriented theory	10
1.3.2 Current Approaches to theory development	10
1.3.3 Approach to theory development in this thesis	12
1.4 Thesis outline	13
2. RELATED RESEARCH	15
2.1 The Symbolic Approach to Artificial Intelligence.....	16
2.1.1 Newell and Simon	19
2.1.2 The SRI International Robot, "Shakey"	21
2.1.3 Stanford Cart	27
2.1.4 The Multiple Autonomous Undersea Vehicles (MAUV) project	30
2.1.5 The NAVLAB project at Carnegie-Mellon	33
2.1.6 The University of Maryland System.....	38
2.1.7 Ronald C. Arkin's AuRA architecture	43
2.1.8 Other Systems.....	46
2.2 The Connectionist Approach to Artificial Intelligence.....	50
2.2.1 The Society of Mind	51
2.2.2 The Robots of R.A. Brooks.....	52
2.2.3 ALVINN, The Carnegie Mellon Steering Network	54
2.3 Summary.....	56
3. THEORY AND THE EXPERIMENTAL IMPLEMENTATION	58
3.1 The Theory.....	59
3.2 The Experimental System.....	63
3.2.1 Terminology	64
3.2.2 Plan-and-Monitor.....	67
3.2.3 Perceptual Servoing: A different perspective on Plan-and-Monitor	71
3.2.3.1 Action-Level Servoing.....	72
3.2.3.2 Plan-Level Servoing.....	73
3.2.3.3 Goal-Level Servoing.....	74
3.3 Outline of Chapters 4, 5 and 6	74

4. REPRESENTATIONS	76
4.1 Representation of the Environment.....	76
4.1.1 The Network.....	77
4.1.1.1 Representing Volumes as Locales	78
4.1.1.2 Faces.....	82
4.1.1.3 Regions.....	84
4.1.1.4 Line Segments	85
4.1.1.5 Points.....	87
4.1.2 Localization and local accuracy vs global accuracy	88
4.1.3 Scope and Perspective: The Locale Hierarchy	91
4.1.4 Compiler.....	93
4.1.5 Summary and Comparison.....	96
4.2 The actuator and sensor models	99
4.3 Representing tasks and the current state of the problem.....	102
4.3.1 Goals	102
4.3.2 Milestones	103
4.3.3 Plan Sketches	104
5. SKETCHING PLANS	107
5.1 Sketch-a-Plan	107
5.1.1 A* and planning costs.....	108
5.1.2 The graph, a heuristic function and some terminology	110
5.1.2 The Successor Function	111
5.1.3 The Sketch-a-Plan Algorithm.....	115
5.2 Remarks about planning complexity.....	117
5.2.1 Complexity of plan sketching	118
5.2.2 Total planning complexity assuming a static environment and no execution errors	122
5.2.3 The effect of execution errors and dynamic environments on planning complexity.....	126
5.3 Experimental Results.....	131
5.3.1 Experiment 1	132
5.3.2 Experiment 2	137
5.3.3 Experimental Experience with Planning Complexity in "Typical" Situations.....	143

6. PERCEPTION AND THE EXECUTION OF PRIMITIVE SUBGOALS	148
6.1 Primitive Subgoals	149
6.2 Action-Level and Plan-Level Perceptual Servoing.....	152
6.2.1 Selecting Landmarks	152
6.2.2 Constructing Appearance Templates from the Model.....	155
6.2.3 Matching Appearance Templates to the Data Using Correlation	156
6.2.4 Action-Level Perceptual Servoing	159
6.2.5 Plan-Level-Perceptual Servoing.....	164
6.3 Remarks About Complexity	166
6.3.1 The complexity of landmark selection, template construction and matching.....	166
6.3.2 The complexity of action-level perceptual servoing.....	168
6.3.3 Complexity of plan-level perceptual servoing.....	169
6.4 Goal-Level Perceptual Servoing	170
7. SUMMARY AND FUTURE RESEARCH.....	174
7.1 Artificial Intelligence Research	174
7.2 Applications Research	176
BIBLIOGRAPHY	178

LIST OF FIGURES

Figure

- 1.1. This thesis explores the concept of a fine grained interweaving of reasoning, action and perception..... 1
- 1.2 In designing the experimental navigation system it has been assumed that the "Go Fetch" problem can be decomposed according to the above diagram..... 3
- 1.3. The goal of moving from the robot laboratory to Ted Djaferis' office, illustrated here as a bold line connecting two black dots, is represented in conceptual dependency by the expression: (ptans robot-laboratory ted-djaferis-office). 4
- 1.4. The agent, "Harvey", used in this investigation is a Denning mobile robot, equipped with a black and white TV camera. 5
- 1.5. The experimental system models the environment, the capabilities of the agent and the task to be performed..... 6
- 1.6. Visual sensors are used as a blind man uses his cane: to verify that his model and his reasoning are consistent with reality..... 7
- 1.7. The equipment and languages used to construct the experimental system were standard, commercially available, products..... 13
- 2.1. An agent's interface with the environment is defined by its sensors and its actuators..... 16
- 2.2. The conventional symbolic approach to research in intelligent agents has been to investigate the functional elements of the system separately. 17
- 2.3. The conventional symbolic AI system has viewed the functional elements of Figure 2.2 as macroprocesses, processes which make large contributions to the intelligent process at each invocation..... 19
- 2.4. Newell and Simon's definition of an information processing system..... 20
- 2.5. Experiments with Shakey were conducted in an indoor office environment similar to the example shown here..... 21
- 2.6. Shakey's world was modeled as a set of predicate calculus statements. 23
- 2.7. The architecture of the system used in the Shakey experiments is one of those which established what R. Brooks has called the "traditional approach" [Brooks, 1986]..... 25
- 2.8. A primary goal of the Stanford Cart project was to enable the cart to build a model of its environment, using vision..... 26
- 2.9. The Stanford Cart environment was modeled in terms of points..... 27
- 2.10. Although the details of the Stanford Cart system are very different from the Shakey system, the architecture is very similar..... 28

2.11. One way to compare the Stanford Cart system with the paradigm used in this system is to view it in terms of Figure 1.1.....	29
2.12. Each MAUV planner constructs a complete plan graph.....	31
2.13. MAUV is designed for controlling a number of vehicles, but at the individual vehicle level the design follows the traditional architecture and information flow.....	32
2.14. The envisioned architecture for NAVLAB system is based on the blackboard concept.	34
2.15. The Color Vision module analyzes a color image to determine the location of the road.....	36
2.16. The architecture of the implemented portion of the NAVLAB design.	37
2.17. The ultimate University of Maryland system is seen as consisting of four major modules: Vision, shown in the dotted box, the Planner, the Navigator and the Pilot.....	38
2.18. As the Maryland system is currently implemented, the vision system performs a complete scene interpretation, determining what is road and not road in the image.....	40
2.19. The Maryland system as it is currently implemented begins by interpreting an image and storing the results in the model.	41
2.20. Viewed from the perspective of Figure 1.1, in the implemented portion of the Maryland system reasoning is a microprocess, but vision and motion processes are macroprocesses.	42
2.21. Arkin's AuRA system divides the navigation task into the traditional macromodules: perception, planning and action.....	43
2.22. Arkin's system design followed his interpretation of the action-perception cycle.....	46
2.23. The Martin Marietta road following system follows the conventional macroprocess architecture.	47
2.24. Vision in the Martin Marietta system is a macroprocess.....	48
2.25. The connectionist approach can be characterized as using large numbers simple processing elements each of which may be as simple as a threshold logic unit (a).....	49
2.26. Minsky proposes that intelligence is accomplished by a large number of very unintelligent modules, called agents, organized into agencies which perform specific tasks (a).	50
2.27. Brooks proposes a "layered" architecture.	52
2.28. The implementation of R.A. Brooks' layered architecture, as illustrated by this diagram from [Brooks, 1986], looks very much like a Minsky society.	53
2.29. ALVINN is neural network consisting of three layers: a "retinal" layer, a hidden layer and an output layer.....	55

3.1. In this dissertation an intelligent agent is viewed as a single reasoning module which has direct access to information from the sensors and can directly control the actuators.	58
3.2. As an agent reasons about what it is doing and where it is located, it builds an internal model of what it is doing, where it is and what it might do next.....	59
3.3. Unlike the traditional approach to symbolic AI, the RML system implements intelligent behavior as three microprocesses, each of which does only what is necessary to perform the next microprocess.	60
3.4. The environment of the story about the boatman and the savage (a) might be perceived differently by the boatman looking for his oars (b), the savage who is looking for firewood (c) and the jogger who had no interest in them (d).....	62
3.5. The experimental system addresses navigation in and near the University of Massachusetts Research Center.....	64
3.6. The interweaving of perception, action and reasoning in the experimental system is orchestrated by an algorithm called "plan-and-monitor", summarized here as a flowchart.....	67
3.7. Plans are developed incrementally.	68
3.8. The first refinement of the goal of Figure 3.5 results in the goal decomposition tree shown here.....	69
3.9. Whenever the first subgoal of plan sketch is not a primitive one it is further refined into a plan sketch which will accomplish that subgoal.....	70
3.10. Plan refinement continues in a depth first fashion until the first subgoal is a primitive one.....	71
3.11. When primitive actions are executed in an open loop fashion they rarely are executed as intended.....	72
3.12. Although action-level perceptual servoing improves the accuracy of each primitive action, it is still possible for errors to accumulate over a series of actions.	73
3.13. Action and plan level servoing work together to control accuracy over complex paths, but if the agent were to encounter a bump, if its actuators were to produce a large erroneous motion or if there were some reasoning error it would be possible for the agent to get lost.....	75
4.1. Locales are implemented as frames with slots which organize the topological, geometric, and physical properties of the spatial entities they represent.	79
4.2. In the experimental system locales are organized into a contained-by hierarchy in such a way that the offspring locales partition their parent.	80
4.3. Part of hierarchies, such as this one, are used to describe objects for recognition purposes.....	81

4.4. Locale surfaces are described in terms of planar faces which, like locales, are represented as frames.....	83
4.5. Faces, like locales, are organized into a contained-by hierarchy.....	84
4.6. The appearance of each face is represented by regions of uniform visual properties.....	85
4.7. Each face and region boundary is represented as a list of connected boundary components which are, in turn, represented as a list of 3D-line-segment frames.....	86
4.8. The endpoints of 3D-line-segments are 3D-point frames.....	88
4.9. A place is represented by a location, which is a frame identifying a locale containing the place and the coordinates of the place as expressed in the local coordinate system of that locale.	89
4.10. Defining location in terms of a locale expresses a degree of knowledge about where a place "is".....	90
4.11. The pose of a 3D entity is represented by a frame which identifies its coordinates and its orientation with respect to the coordinate system of a locale.	90
4.12. The contained-by hierarchy offers ways to express scope and perspective when reasoning about space.	92
4.13. The first step in describing a locale is to specify the overall structure of the locale and its relationship to its parent.	94
4.14. Once a the overall structure of a locale has been defined, details can be specified by partitioning the faces into smaller subfaces.	95
4.15. The appearance of a face is described in by breaking it into regions of uniform appearance.....	96
4.16. The model of the environment is incomplete, but locally positionally accurate.....	97
4.17. Kuipers represents space as a network of nodes, representing places.....	98
4.18. Lawton, Levitt, McConnel, Nelson and Glicksman used an is-a hierarchy as part of their navigation system.	99
4.19. For Harvey the actuator model is relatively simple.....	100
4.20. Landmarks are 3D structures which have physical properties which make them distinct from other 3D structures nearby and which are locally unique.....	103
4.21. Milestones are used to check to see if reasoning and reality are consistent.	104
4.22. The first step towards achieving a goal is to decompose it into subgoals in a depth first fashion until the first subgoal is a primitive one.	105
4.23. As each subgoal is accomplished it is removed and the next subgoal is refined in a depth first fashion until the new first subgoal is primitive.	106

5.1. The A* Algorithm summarized from [Nilsson 1980].....	109
5.2. All spatial volumes are represented as locales, emphasizing the fact that they can at times be an obstacle and at other times be the environment (a).	110
5.3. The extent of the plan sketching space is determined using the contained-by hierarchy.....	112
5.4. Successors to a start location, S, are determined in part by the situation.....	113
5.5. The successor function algorithm.....	114
5.6. The sketch-a-plan algorithm.....	115
5.7. The first step in sketching a plan is to chose a scope locale.....	116
5.8. In some situations the structure of the initial choice for a scope-locale does will not contain a solution path.	117
5.9. In the simplified model of the University of Massachusetts environment locales correspond to areas such as lawn, buildings, floors and rooms.....	119
5.10. The locale hierarchy corresponding to the simplified model of the University of Massachusetts (illustrated in Figure 5.9) shows how the environment is partitioned into sublocales.	120
5.11. As knowledge of the environment becomes more detailed or expands to other parts of the universe, the contained-by hierarchy grows	121
5.12. The number of path segments, ρ_A , produced by A* for a given goal sometimes differs from the number of path segments, ρ_M , produced by plan-and-monitor for that same goal.....	123
5.13. By using the methods outlined in the text when dividing locales into sublocales, the effect that the division has on path length can be controlled to the extent $O(\rho_M)=O(\rho_A)$	125
5.14. The RML architecture develops lazy plans, deferring most of the computation until it is required.....	128
5.15. The planning experiments were conducted using a locale network model of this portion of the University of Massachusetts Research Center (LGRC).....	130
5.16. Given the goal of moving from the start-point in the robot-lab to the goal-point just inside the office, plan-and-monitor invokes sketch-a-plan which returns a plan sketch.....	133
5.17. The first subgoal of figure 5.16 was not primitive because objects are in the way.	134
5.18. When the primitive goals of figure 5.17 have been completed and Harvey has passed through the door, the next segment of the journey is refined.	135
5.19. When the hallway path has been traversed and Harvey has passed through the doorway to the office, the details of the office path are developed and executed.	136

5.20. The solution to the problem of getting from the robot lab to the south east lobby begins with a rough plan sketch.	137
5.21. As with the robot-lab to office problem, the robot-lab portion of the trip is refined before the journey begins.	138
5.22. Once in the hallway it is necessary to avoid the doorjam, but this time the remainder of the trip is a primitive subgoal.	139
5.23. Since no obstacles were present in the south west lobby the subgoal corresponding to this portion of the trip was already primitive.	140
5.24. Traveling through the lobby meant avoiding the stairwell, which protrudes into the lobby.	141
5.25. The final subgoal is primitive and brings Harvey to the desired location.	142
5.26. The goals used in the complexity experiments described in section 5.3.3 are ptrans expressions using the locations pk identified above.	143
5.27. These results summarize planning complexity experience when the plan-and-monitor algorithm was used.	144
5.28. The number of nodes expanded as a function of path length during the experiments performed using plan and monitor did not exhibit exponential growth for path lengths less than 20, the longest path attempted.	145
5.29. When planning was done using the A* algorithm using only the model for the robot-lab the results were quite reasonable, but the measured complexity was higher.	146
5.30. Expanding the model to include the second-floor of the Research building increased the measured complexity for the A* algorithm considerably.	146
5.31. As predicted by the analytical arguments of Section 5.2, the number of nodes, N, expanded as a function of path length grew far more quickly for A* than for the plan-and-monitor algorithm.	147
6.1. If there is an unobstructed straight line path between two locations location-1 and location-2 then the subgoal (ptrans location-1 location-2) can be accomplished by turning toward location-2 and then moving forward the distance d.	148
6.2. Achieving navigational goals is the result of three levels of perceptual servoing: action-level, plan-level and goal-level.	151
6.3. Distinctive reflectance patterns are found where regions of differing reflectances meet, namely at boundary segments (a, b) and vertices (c).	153
6.4. Many distinctive reflectance patterns can be found at vertices. The vertices which can be seen from inside a locale can be found using the network.	154
6.5. The reflectivity pattern surrounding each vertex can be found by retrieving the reflectances of the regions associated with that vertex.	155

6.6.	Before each action begins, landmarks are selected (a) using knowledge of the environment and the task. Action is then carried out incrementally.....	159
6.7.	Starting at the location marked by the symbol an agent will deviate from its intended line of motion and finish an incremental motion at the location marked by the symbol . Experiments have verified that, for very small distances, the path between.....	160
6.8.	Once the heading error has been determined, the robot is turned toward a distant point on its original intended path.....	162
6.9.	A simulation of the action-level perceptual servoing concept shows excellent control when the camera is perfectly aligned to the direction of robot motion ($\Theta = 0.00$).....	163
6.10.	Using action-level perceptual servoing dramatically improves the ability of the robot to remain on course.....	164
6.11.	Using correlation for landmark matching, together with 3D pose determination experiments have shown the ability to determine the robot's location to within 1.5 inches.....	165
6.12.	The "Where am I?" function required for goal-level perceptual servoing is seen as using the contained-by hierarchy (a) to guide a search of the network for the locale whose free space sublocale contains the agent.....	171
7.1.	Model building, obstacle detection and recovering from being lost have been left as issues for future research.	176
7.2.	The experimental system described in Chapters 3-6 make use of a CAD-like model, together with the RML strategy to address problems in navigation in an efficient way.....	177

CHAPTER 1
INTRODUCTION

One chilly evening two people, a boatman and a savage, came upon two long narrow pieces of wood on the shore. The savage, anxious to warm himself by a fire, immediately recognized the objects as pieces of firewood; the boatman, looking forward to returning home to the mainland, had no trouble seeing them as his missing oars. After convincing the savage of his need for the objects the boatman returned to his wife, who had been waiting by the rowboat, and the two of them set off for the mainland. As they approached the mainland shore the wife, pointing to a building in the distance, remarked "It looks like the mayor has had his house repainted". For the first time the boatman became aware that the landmark he had been using for steering was a house. Noting the position of the sun he decided that it would be rush hour when they landed and he began thinking about how to avoid traffic on the way home.

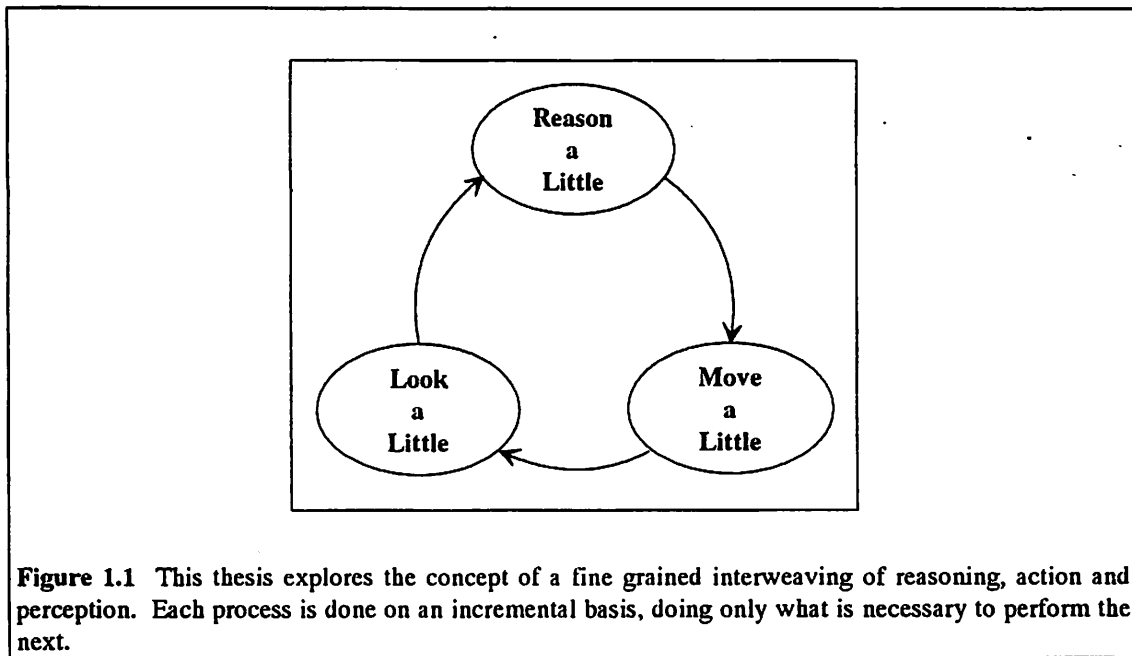


Figure 1.1 This thesis explores the concept of a fine grained interweaving of reasoning, action and perception. Each process is done on an incremental basis, doing only what is necessary to perform the next.

As this story illustrates, our perceptions are affected by our knowledge, by what we are thinking about and what we are doing. In turn, what we know and what we are thinking about and doing is affected by our perceptions. This thesis is concerned with this interaction between reason, action and perception. It takes the position that intelligent behavior can be demonstrated and efficiently executed

using traditional Artificial Intelligence techniques by introducing a fine grained integration or "interweaving" of these three activities as illustrated in Figure 1.1. By performing these activities incrementally, doing only what is necessary to begin the next process, the system as a whole becomes more efficient because questions are posed and answered in a timely fashion and because the questions are more focused, making it easier for the next process to do its job. The overall effect is to simplify all three activities, making them easier to understand, to implement, and to compute. This claim is supported in the following chapters which reveal the detailed design and explore the effectiveness of an experimental system which implements this idea in a specific application area: autonomous mobile robot navigation.

In addition to asserting and supporting the central claim, the work described in this dissertation has been directed towards a second goal: to contribute to the field of vision by developing the experimental system in such a way that it addresses the need for a vision theory for industrial applications. As will be discussed in Section 1.3, vision has not been applied as broadly in industry as was hoped in the 1970's and 1980's because the theory of vision is not yet mature enough to support *routine* engineering of applications. This thesis contributes to that maturity by developing a system which requires only standard computing equipment, addresses real time issues on this kind of equipment and illustrates the solution to some difficult problems in a simple, broadly applicable way. This claim is discussed in considerably more detail in Section 1.3.

The remainder of this chapter begins by describing the navigation problem addressed by the experimental system and places it in the context of a larger problem which provides a number of future research problems. This is discussed in Section 1.1. Section 1.2 follows with an outline of the approach used to implement the "interweaving" architecture and Section 1.3 discusses the application problem and discusses how the work in this thesis addresses the need for such a theory.

1.1 The Problem - Navigation

The problem area selected for investigating the interaction between planning, action, and perception is that of controlling a mobile robot in performing intelligent tasks. This area is well suited to the goals of this thesis for three reasons: it offers a natural setting for working with reasoning, action and

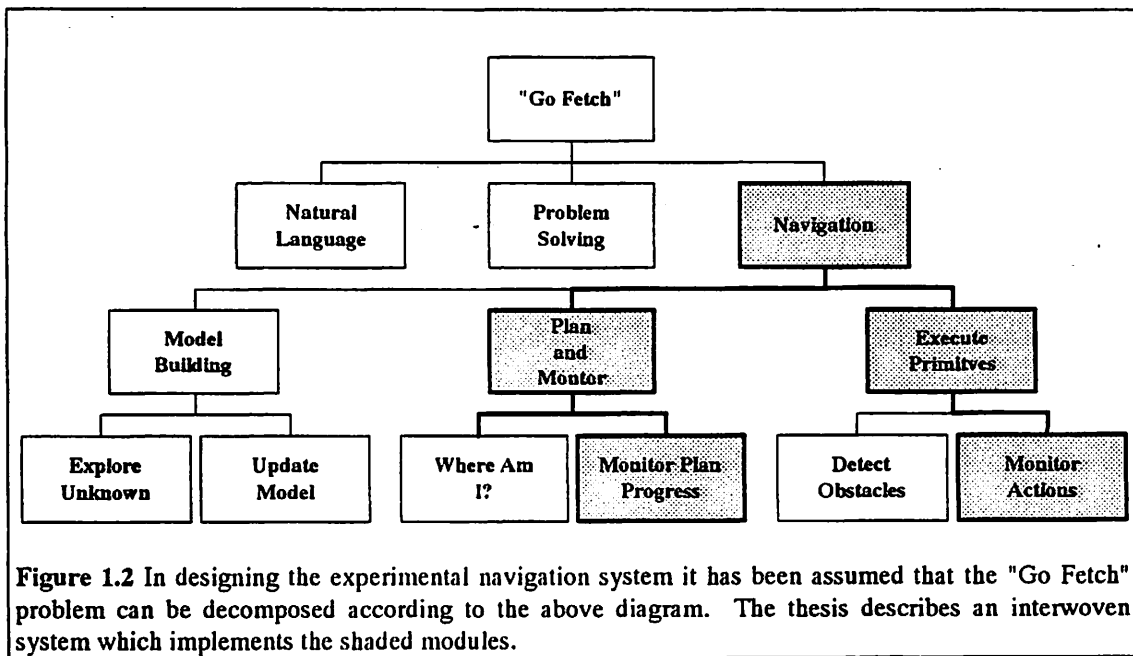
perception by providing a rich variety of problems which can test this idea; it provides a good testbed for ideas which must work under a variety of environmental conditions; and it is a recognized application area, sharing many of the technical problems found in industrial applications. The work described here has been directed at the navigation part of what shall hereafter be called the "Go Fetch" problem. In this problem, descriptions of the environment are given to the agent, some of which may be entered via speech or natural language statements, such as

"There is a red book in Ted Djaferis' office."

Then, given a command such as:

"Go get it."

the agent should find its way through the environment to locate the book and return with the book.



It is assumed in this thesis that this problem is decomposable and Figure 1.2 shows what are seen as the principal subproblems in this decomposition. In general, the levels in the figure correspond to levels of abstraction and processing *nominally* moves from left to right (this is not strictly true since, as has been indicated, this investigation assumes there is a considerable amount of interweaving of the "subproblems"). Thus, the Natural Language module would convert input sentences such as the command "Go get it." into a set of goals, which are represented in a manner which follows closely the

work of Schank and Abelson on conceptual dependency [Schank and Abelson, 1977]. The above sentence, for example, would result in the goal

(and
 (ptrans laboratory ted-djaferis-office)
 (atrans book from djaferis to agent)
 (ptrans ted-djaferis-office laboratory)
 (atrans book from agent to fennema))

where ptrans represents a physical transfer of location and atrans a transfer of possession. These goals would then be passed to the problem solver, which among other things, would pass navigational goals such as the goal:

(ptrans laboratory ted-djaferis-office)

to the navigation module (Figure 1.3). It is this navigation subproblem which is the problem addressed by the experimental system.

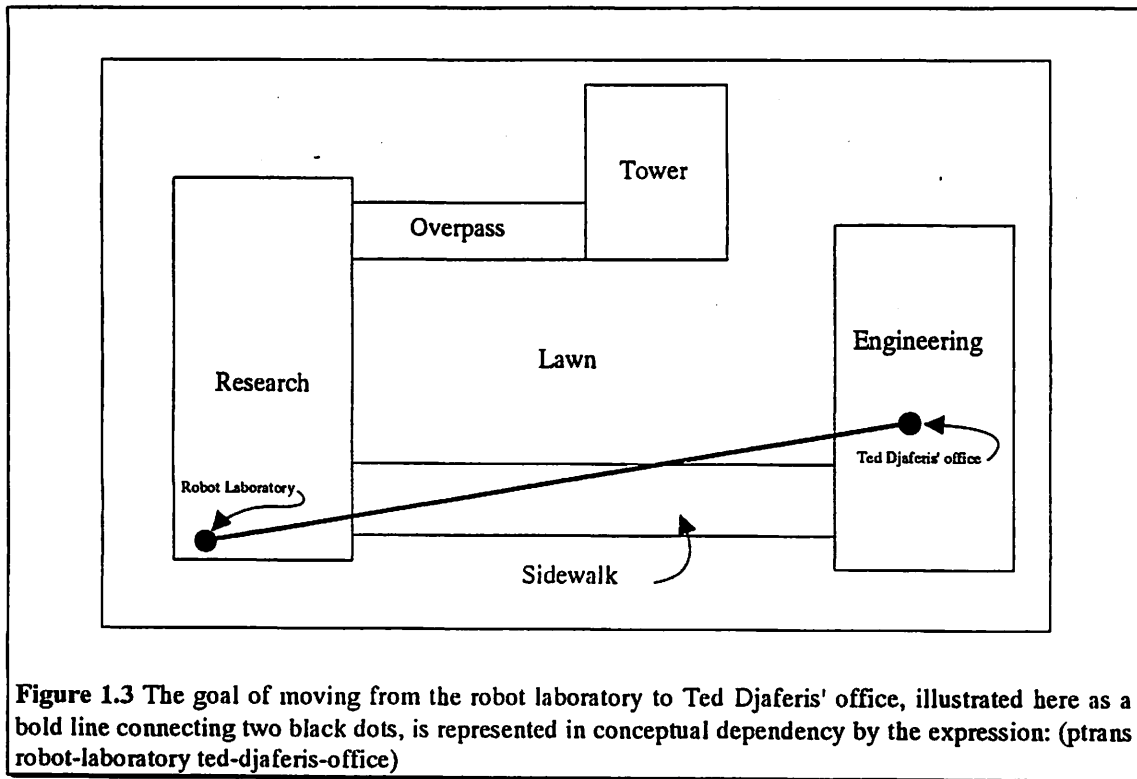


Figure 1.3 The goal of moving from the robot laboratory to Ted Djaferis' office, illustrated here as a bold line connecting two black dots, is represented in conceptual dependency by the expression: (ptrans robot-laboratory ted-djaferis-office)

Navigation uses vision for several purposes. The bottom row in Figure 1.2 identifies the vision capabilities seen as required by a complete navigation system. The experimental system described in this work does not implement all of these capabilities. It has been assumed that a partial, but accurate model

exists, that no objects are unknown, and that the agent is the only moving object. This dissertation will therefore not address model building or obstacle detection. Spatial representations have been investigated, but the data used in those representations has been entered by hand.

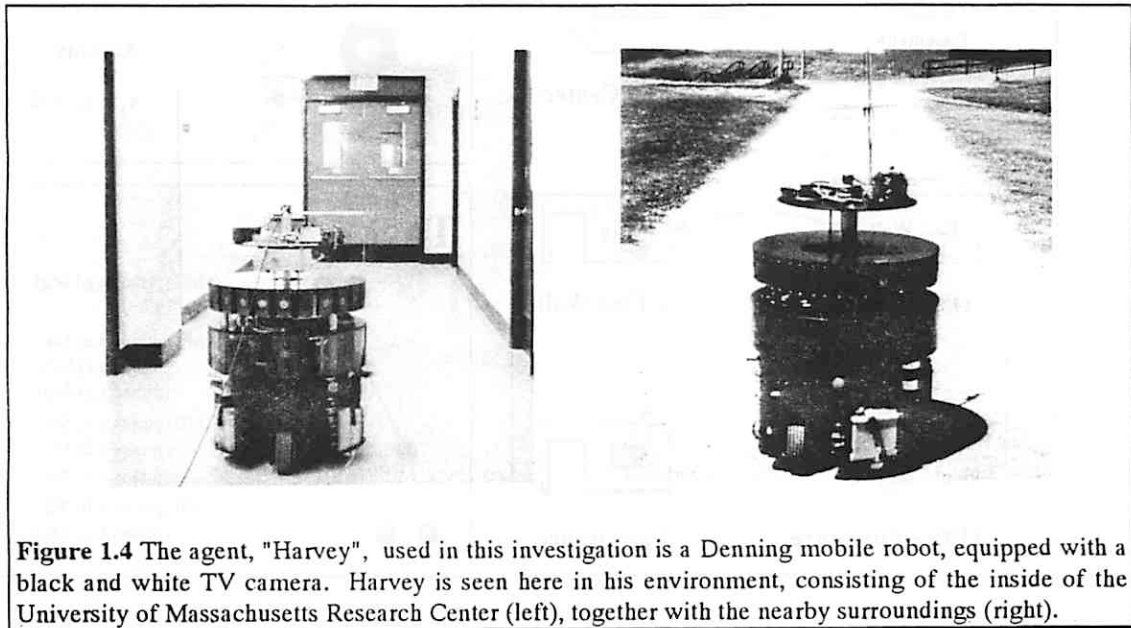


Figure 1.4 The agent, "Harvey", used in this investigation is a Denning mobile robot, equipped with a black and white TV camera. Harvey is seen here in his environment, consisting of the inside of the University of Massachusetts Research Center (left), together with the nearby surroundings (right).

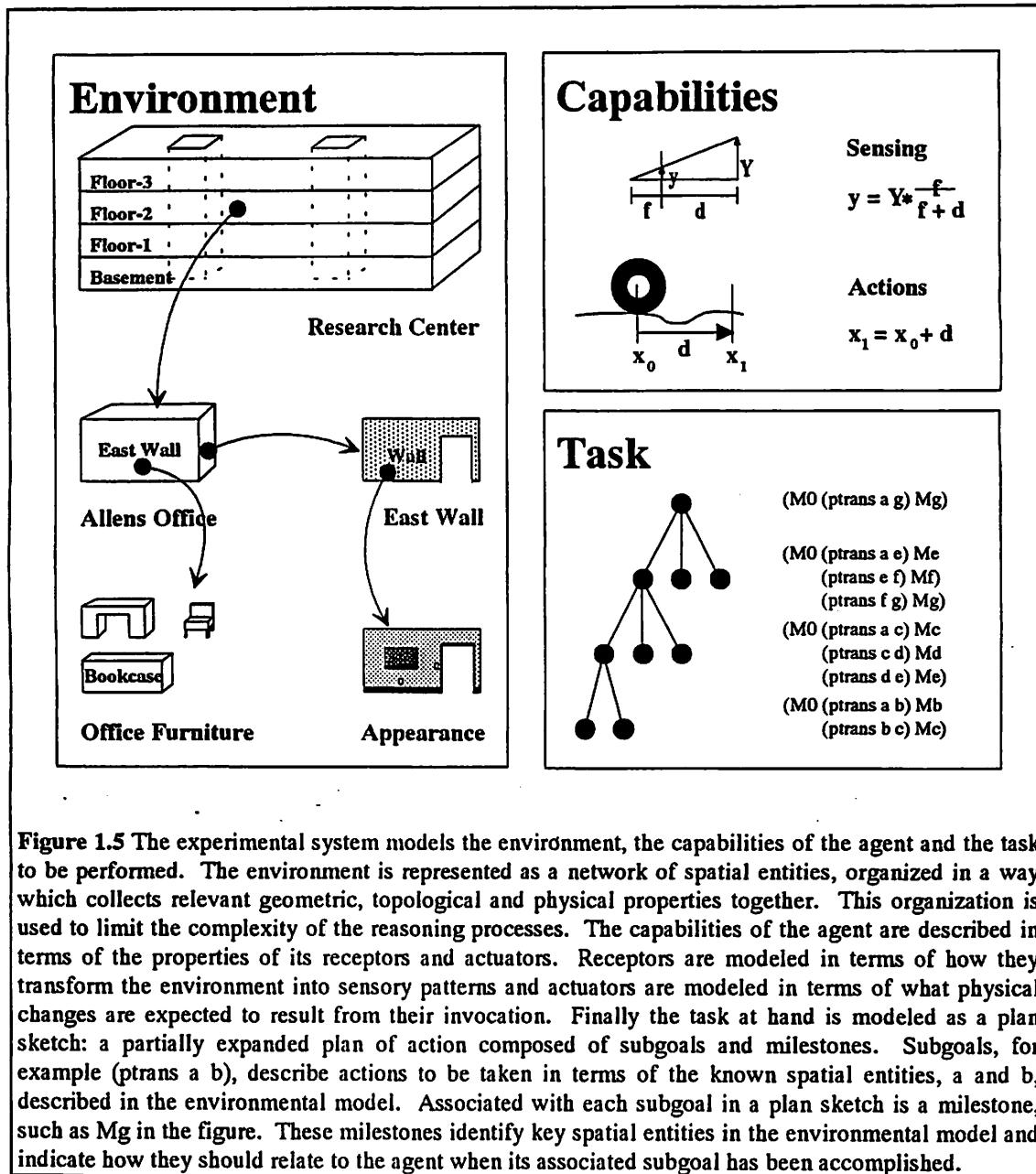
In this context the problem addressed by the experimental system can be more precisely stated by specifying the agent (the sensors, effectors and processing hardware), the environment and the behavior to be produced:

The agent, "Harvey", is a Denning mobile robot equipped with a black and white 512 x 512 TV camera, both connected via an umbilical cord to a minicomputer. (Figure 1.4)

The environment consists of a portion of the University of Massachusetts Lederle Graduate Research Center. During experiments it is assumed that there will be no moving or unknown objects. It is further assumed that all descriptions of the environment are given to the robot in the form of a model, described in Chapter 4.

The behavior to be produced is: given a navigational goal, stated in conceptual dependency terms as (ptrans a b), the agent should find its way through the environment from location a to location b.

This work described in this dissertation has developed representations, control structures, planning mechanisms and vision capabilities sufficient to demonstrate the central claim of this thesis in the context of this problem.



1.2 The Approach

The implementation of the ideas described above begins with a central model. This model contains information about the environment, the robot's capabilities, and the task at hand (Figure 1.5). The environmental model contains information about the geometric, topological and physical properties of space. The capabilities model captures the effects of sensing the environment with the camera and of attempting to take actions. These two portions of the model provide the information necessary to reason

about routes, motion and landmarks which may be seen along a journey. As will be described in Chapter 4, the environmental model is organized into a network which offers focusing mechanisms to control the complexity of the reasoning processes. The task model represents the goal, the current view of how this goal will be achieved and what sensory events will be used to decide whether or not the agent is making progress.

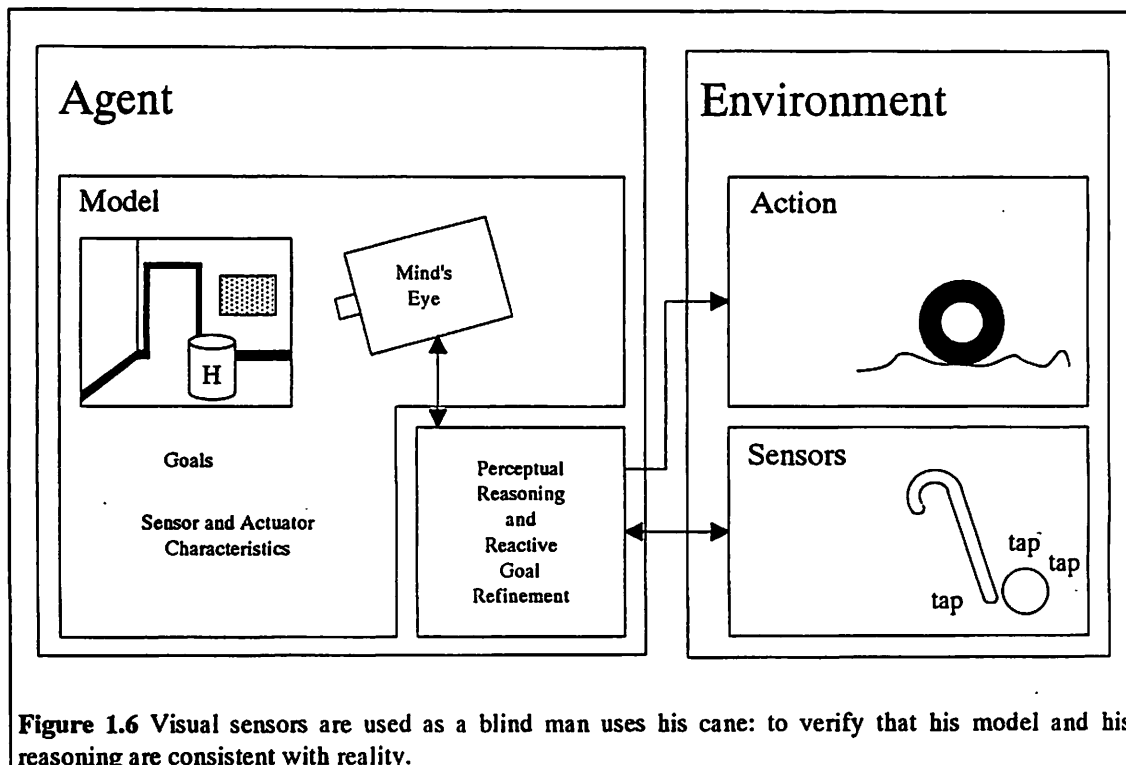


Figure 1.6 Visual sensors are used as a blind man uses his cane: to verify that his model and his reasoning are consistent with reality.

The experimental system implements a version of the loop of Figure 1.1. Beginning with a goal the system reasons about a course of action, doing only what reasoning is necessary to decide what move to make next. During this reasoning increment the system predicts how its perceptions should change as a result of this action. The action is then executed and the perceptual expectations are compared with what is seen after the action. The results of this comparison influence the next reasoning increment. Each reasoning step makes use of the task model, the environmental model, the capabilities model and the latest sensory input to decide what next action to perform in order to make progress toward the goal.

Perception (vision) in this system does not take on the flavor of an image understanding system or a scene analysis system. Rather perception is seen as a means for checking to see whether or not reasoning

and reality are consistent. The visual sensor is used in a way which is quite analogous to the way a blind man uses his cane (Figure 1.6). The agent is viewed as choosing its actions based on reasoning done with its model and using its visual sensors for evidence that its model and "reality" are in agreement.

This approach, and its implementation as described in the following chapters, is novel. Its implementation and the results of experiments with the ideas contribute to the fields of Artificial Intelligence and Vision in the following ways:

1. The investigation has posed and demonstrated some novel AI and Vision principles:

a) Intelligent behavior can be demonstrated and efficiently executed using traditional Artificial Intelligence techniques by introducing a fine grained integration of reasoning, perception and action. This has been accomplished by the construction of a novel system which uses a central model and which interweaves these processes in a control structure which uses a fine grained process integration to bring the appropriate knowledge to the problem at a time when it is most useful. Demonstration of the navigation portion of the "go fetch" task has shown that the architecture can emulate an intelligent behavior. The efficiency of the system has been demonstrated by its near "real time" performance.

b) Vision tasks can be efficiently and effectively implemented in a "top down" manner. This has been accomplished by treating vision, not as a subsystem which itself has great powers, but rather as part of a reasoning activity which uses sensors to probe the environment in order to verify that its reasoning and reality are consistent.

c) Vision can be made simpler by its "integration" into the system, rather than treating it as a separate subsystem. This architecture makes it possible to use the visual sensor as a means for answering highly focused questions in a timely fashion, rather than expecting a vision system to guess which questions may be asked. Vision systems which must "interpret" an image without this kind of guidance may interpret it slowly and, as illustrated in the story at the beginning of this chapter, may interpret it inappropriately.

d) Vision tasks can be accomplished with a simple "low level" operator. The issue is not which operator to use, but what knowledge can be applied.

2. Current models of intelligence divide the problem into subproblems each of which is NP complete and which may give the wrong answer due to lack of communication. Vision, in particular, is asked to interpret scenes without knowing why. This system redistributes computational effort in such a way as to indicate how we can build vision systems which solve "difficult" tasks in real time with conventional computing power.

These claims will be supported in the chapters which follow, beginning with Chapter 2. The remainder of the current chapter addresses the second goal of this thesis: that it contribute to the development of an applications-oriented theory of computer vision. That discussion begins with a more detailed motivation for this need.

1.3 Applications

Although interest in computer perception was originally inspired primarily by military applications, it soon became apparent that there were numerous potential commercial uses for the technology. The military interest of the 1940's and 1950's was augmented in the late 1950's by the perceived commercial need for Optical Character Recognition (OCR), now used for sorting bank checks and reading price tags at department stores. In the mid 1970's the list of promising applications had grown to include the automation of differential blood cell analysis, visual inspection of parts on an assembly line, visual navigation of autonomous vehicles, visual guidance of robot arms in assembly tasks and many other tasks of a similar nature. By 1980 there were a number of success stories and the total market for commercial applications of computer vision was thought to be in the tens, even hundreds of billions of dollars. A look at the commercial computer vision business in 1990, however, shows that it remains to be orders of magnitude below those estimates. Why?

The basic reason is simple. The field of computer vision has not yet matured to the point where useful application systems can be economically engineered on a *routine* basis. Four factors contribute to this situation:

1. The state of the art in computer vision is such that each application development entails a considerable research component. This component makes routine application development too risky. There is a need for an application-oriented theory which better teaches how to build and engineer vision systems.
2. Applications have demanding real time requirements. The current state of the art in computer vision cannot address many applications without expensive, special purpose hardware. There is a need to address real time computation of vision results.
3. The current theory is not sufficiently robust or general that engineering from one application can be applied economically to another. There is a need for more generality at the application level.
4. Relatively few people are well enough trained to make use of the technology which exists today .

The work in this thesis has been directed towards contributing to the first three of these factors. The next two sections describe the application situation in more detail and Section 1.3.3 indicates how this thesis has addressed these needs.

1.3.1 The need for an application oriented theory

Every successful computer vision system has required the use of special lighting, has had a very narrow application and/or has demanded that the objects be redesigned to make the problem simpler. When Optical Character Recognition systems were developed, for example, the extant vision theories of the time (pattern recognition) did not offer a solution. The theory was only mature enough to indicate that the problem could not be solved without restricting the input to a carefully designed alphabet (OCR-A and OCR-B). Solutions such as this are worthy of high praise indeed, but it is not often that the resulting vision systems offer much to be transferred to other applications. Consequently, it is typical that each application system must be developed "from scratch", usually resulting in costly and risky engineering efforts which often fail or elude funding. The lack of an application-oriented theory of vision makes development so costly that many of the applications included in market estimates cannot yet be developed. What is needed is a body of organized knowledge which better reveals how to solve such problems or, better yet, a general system which works in a very broad class of situations.

1.3.2 Current approaches to theory development

These application needs are currently being addressed from different perspectives by four groups. The first group, the vision research community, seeks to understand how to build the general system. In the decades that have passed since the 1950's this community has pursued this goal following a number of different philosophies. Larry Roberts [Roberts, 1965] developed the first vision system philosophy. His experiments showed that his ideas were capable of (often) identifying the shape of white blocks on a black background. Since that time [Brooks, 1981a, 1981b], [Hanson and Riseman, 1978, 1987], [Marr, 1980], [Barrow and Tenenbaum, 1982], [Ballard, Brown and Feldman, 1978], [Nagao, Matsuyama and Ikeda, 1978, 1979], [Ohta 1980], [Shari, 1978], [Rubin, 1978], [Matsuyama and Hwang, 1985], [McKeown, Harvey and McDermott, 1985], [Mulder, Mackworth and Havens, 1988], [Matsuyama, 1987], [McKeown, 1988], [Herman, Kanade and Kuroe, 1984], [Tsotsos, 1985] and others have proposed models of the vision system which hope to extend the capabilities of computer vision to less artificial scenes. These models have formed the research philosophy for the vision community. Their critical components have given rise to the commonly accepted issues in the field and from the investigation of

these issues the field has made a considerable amount of progress in understanding image related processes such as: line detectors, region growing algorithms, interest operators, shape from shading, perceptual grouping, representations, and object recognition algorithms. These models, however, represent research strategies, not working systems. They do not yet provide the answer. They hope to explain the powerful capabilities of human-like perception but, in their present form, are complex, still controversial and in many cases seem to require processing hardware which is currently far from commercially available. Understanding these models and what can currently be used from them requires a person with a great deal of experience in computer vision. Consequently, they currently do not serve as useful guidance to the typical application engineer, who lacks this experience.

A second group has interpreted the general theories of the research community and developed the most general vision system the current theory will support. This has resulted in vision modules or systems, beginning with the SRI vision module [Gleason and Agin, 1979] and leading to the formation of companies like Machine Intelligence Corporation and Automatix. These vision modules were capable of recognizing and locating simple objects, but they were not sufficiently general or tolerant of environmental changes to be widely applicable. The general theory is still too embryonic to foster a useful general system.

The third group has been developing applications using an "anything goes" approach. Most of the successful "vision" applications, like the OCR application mentioned above, have fallen into this category. Problems which arise due to a lack of knowledge about vision are often solved using methods other than "pure" computer vision. This has resulted in the use of laser range finders, such as in the Carnegie Mellon ALV project [Thorpe, Herbert, Kanade and Shafer, 1988], structured light such as was used in CONSIGHT [Holland, Rossol and Ward, 1979], and restructuring of the objects to be recognized as was the case for OCR and in the General Motors circuit chip inspection system [Baird, 1978]. The vision module companies soon found that using this "anything goes" approach was the only way they could survive and new companies, such as Diffracto, were formed. This approach has lead to a great deal of useful technology, but it is very often not computer vision technology. The focus for this group tends to be on getting the next application done, rather than generalizing vision.

The fourth major group has designed and brought to market vision development systems such as INTERSYS-XAMINE [Control Data, 1983,1984], Powervision [Riley, McConnel and Lawton, 1988], Imagecalc [Quam, 1984] and KB-VISION [Amerinex, 1989a, 1989b], [Williams, 1990]. The hope for these systems was to make the development of special systems more efficient. It has been discovered, however, that they only ease the situation when placed in the hands of an experienced vision expert. These systems have been quite useful for teaching vision and have served as useful productivity improvement tools in the research community itself, the home of these experts. They have been of less value to the applications community where such experts are rare. So, the research community focuses on the difficult generalization problem and, because of the difficulty in bridging the technology gap, the applications community defaults to alternative technologies. A theory or body of knowledge which reveals how to use vision in the specific applications of today continues to be absent.

1.3.3 Approach to theory development in this thesis

Like the third approach above this thesis has focused on a specific application; but, unlike that approach, it has restricted itself to using vision as the only means for sensing the environment. In addition, this project has taken a look at vision in the context of the entire application system. It has sought to find simplifications to the vision process by considering the system as a whole, looking closely at what can be learned by studying the interaction between components which, to one degree or another, are part of every application system: planning, action, and perception. Vision is not viewed as an isolated subsystem; but, rather, as a function which is tightly interwoven with the overall problem solving process in such a way that the visual sensor is used to make simple queries about the environment.

This approach to theory development has resulted in a novel system design as well as a number of novel representation, planning and vision concepts. This work contributes to the development of applications in the following ways:

1. It shows, at a practical level, how vision can be used to solve application problems. It does this by describing the construction of a complete system. The description of this system is instructive since:
 - a) Vision is used to solve two problems: vehicle steering and monitoring plan progress. These solutions are illustrative of how vision can be used in application settings.

b) Vision has been implemented in a way which can cope with shadows and with variations in scene illumination. The design details show how simple vision techniques can be used to make vision systems which are tolerant to environmental conditions.

c) Vision has been based on a single low level algorithm, embedded in a complete working system in such a way as to demonstrate how many forms of knowledge can be used to simplify real vision problems.

2. Vision is accomplished using facilities which are typical of what can be found in or easily acquired for the application environments of today. The vision related facilities used in the work described in this thesis used conventional sequential computing hardware, commonly available programming languages and moderately priced sensing equipment (Figure 1.7 lists the specific equipment used). This has ensured that the ideas developed can be transferred to an industrial environment.

3. The system design makes heavy use of a representation scheme which is an enhancement of a CAD-like model, building on an existing industrial tool. The result is a system that begins to use CAD information for planning, vision and action.

4. The completion of this system is a step toward the design of a general purpose vision system. Vision is model driven, using a structured and annotated CAD model which forms the basis for representing things which will be seen. New objects can be added by describing their appearance (in a simple manner) and their location (in some room or area). This technique is similar to what is envisioned for an industrial environment.

Component	Description	Period of use
Computing Hardware:	Digital Equipment Corporation Vax 11/750	Early Experiments
	Sun 4	Final Experiments
	AST 486/33	Final Experiments
TV Camera	Sony	All Experiments
Languages	COMMONLISP and C	All Experiments

Figure 1.7 The equipment and languages used to construct the experimental system were standard, commercially available, products. No special hardware was used.

1.4 Thesis outline

The next few chapters describe the system as it was implemented and relates it to other work. Chapter 2 discusses related research as it applies to the overall system architecture. Chapter 3 describes the theory and the overall architecture of the system in more detail. Chapters 4-6 describe the details of

the system including: the representations used, incremental planning and control and perception, respectively. Results are included in each of these chapters, as appropriate. Chapter 7 summarizes the results and describes future research directions.

CHAPTER 2

RELATED RESEARCH

The principal claim of this thesis, as discussed in chapter 1, is a system level issue. This claim addresses the nature of the decomposition of the intelligent process into subprocesses and how these subprocesses should interact. The two primary models for this in today's literature are exemplified by work in the symbolic AI community and the connectionist AI community. The general practice in the symbolic AI community has been to break the intelligent process into macroprocesses called perception, reason and action, each of which performs a complete process in a sequence: perceive, reason, act. The connectionist AI community, on the other hand has divided the process into micromodules, claiming that the intelligent process is a very distributed one. As was described in chapter 1, this thesis takes the position that intelligent behavior can be demonstrated and efficiently executed using a symbolic approach to Artificial Intelligence by introducing a fine grained interweaving of reasoning, perception and action. It is the goal of this chapter to clarify this claim. This will be done by reviewing and summarizing positions taken in the literature and then restating the claim in the context of this review.

Later chapters will describe a system built to investigate this fine grained, interwoven architecture. In building this system there are a number of subordinate claims which have resulted from viewing intelligence in this way:

Vision can be efficiently and effectively implemented in a "top-down" manner when vision is viewed as a means for testing for consistency between reasoning and reality.

Reasoning can be accomplished more efficiently in this framework as an incremental process, doing only what reasoning is necessary to decide on the next action.

Space should be represented as a network which offers reasoning and perceptual focusing mechanisms. In addition, space should be described locally, rather than globally, to deal with inaccuracies.

It would be confusing to the reader to attempt to discuss work which relates to all these issues in one chapter. Work which focuses on specific vision, reasoning, or representational issues will be more helpful if left for discussion in later chapters. The discussion in this chapter will be limited to work

which has addressed the entire system. Issues of vision, reasoning and representation for these projects will be discussed here as well. This will help set the framework for the discussion in later chapters.

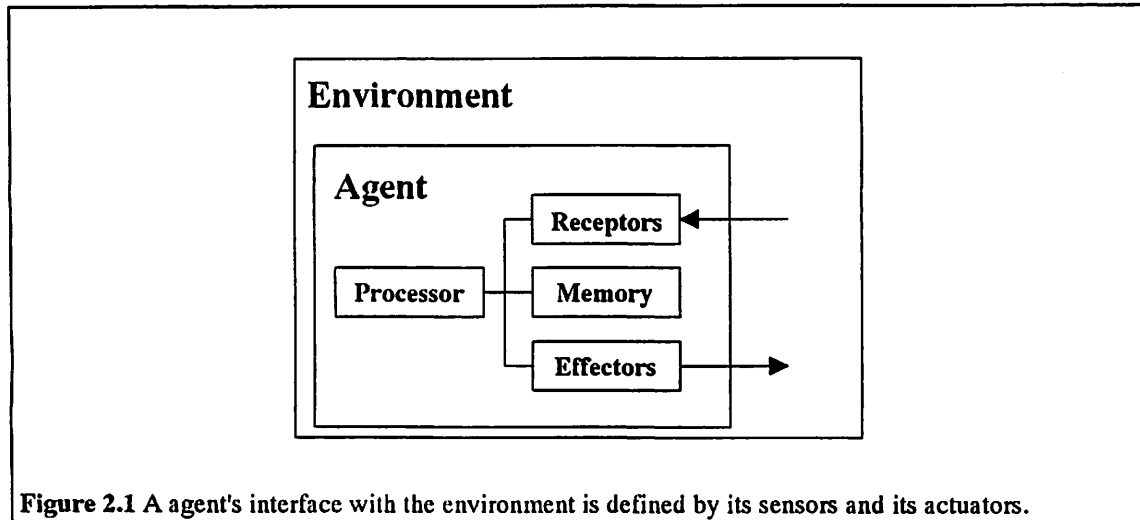


Figure 2.1 A agent's interface with the environment is defined by its sensors and its actuators.

In what follows we will define an agent to be an entity consisting of receptors, effectors, some sort of processing capability and memory (Figure 2.1). The problem of simulating intelligent behavior is first of all to decide what behavior is to be simulated. Once this decision has been made the remainder of the problem is that of identifying the character of each of the agent's modules or devices and then specifying how to orchestrate them so that the agent demonstrates the intelligent behavior. There are a large number of problems in the AI literature which have been characterized as intelligent. System level research, however, has developed around an intelligent robot, usually solving a navigation problem. There are two major approaches to this problem: the symbolic AI approach, and the connectionist AI approach. The symbolic AI approach is described in section 2.1. The connectionist approach is described in section 2.2. Each of these two sections begins with a characterization of the general approach and then describes individual projects. Finally, section 2.3 includes a summary of these approaches and a discussion of how they relate to the central claim of this dissertation.

2.1 The Symbolic Approach to Artificial Intelligence

The symbolic approach to Artificial Intelligence interprets memory (see Figure 2.1) as consisting of structures, sometimes called "symbol structures" or "data structures". According to this view, intelligent behavior arises as a result of processes which produce these structures from data input from the receptors,

modify these structures in response to a set of goals and translate the resulting structures into action via the effectors. The work described in this dissertation falls into this category, but it will be argued that other researchers have explicitly divided their processing into nearly independent "macroprocesses" (Figure 2.2). The term "macroprocess" is used here to emphasize that each execution of the process makes a major contribution to the intellectual process being emulated. One execution of a planning macroprocess, for example, would create a detailed plan. By contrast, the work described in this dissertation divides the processing into "microprocesses". These processes are given the same names, but each process invocation makes only an incremental contribution to the intellectual process. One invocation of a planning microprocess, for example, would sketch a plan only to enough detail to make it possible to decide what move to make next.

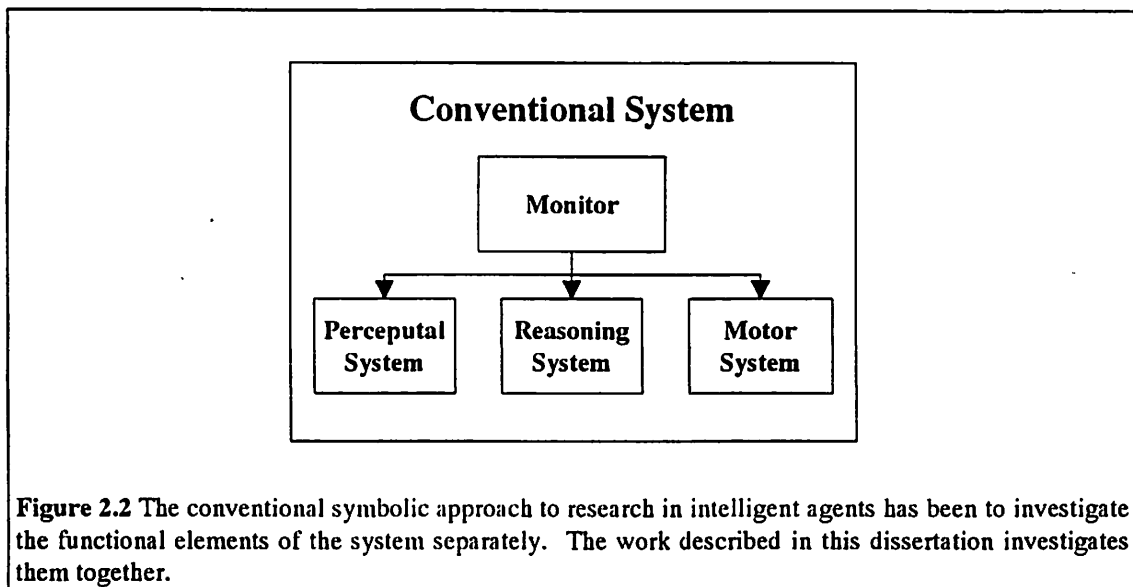


Figure 2.2 The conventional symbolic approach to research in intelligent agents has been to investigate the functional elements of the system separately. The work described in this dissertation investigates them together.

It will also be argued in this section that information flow between these macroprocesses is as depicted in Figure 2.3. Thus, according to the practice of the conventional symbolic AI researchers, represented by the projects described in this section (subsections 2.1.1-2.1.8), the intellectual process is composed of three macroprocesses: perception, reasoning and motor activities. The overall process flow depends on perception which is responsible for interpreting an image and using this interpretation to build or update a model; the reasoning system begins with the model and a goal and constructs a plan of actions to be accomplished; and the motor system carries out the plan.

This flow differs from the paradigm used in this dissertation. Processing in this paradigm is done incrementally and each increment is goal driven. Starting with a goal the reasoning process decides what incremental action to take next and what perceptions should result as a product of that action. Once this decision is made the action is taken. When the action is complete the perceptual expectations are then compared with the actual perceptions. Any differences between expectations and what is sensed are used in the next invocation of the reasoning processes to decide what to do next. The products expected from single invocations of the perception, reason and action macromodules of the conventional symbolic AI approach are developed here in a "reason a little, move a little, look a little" (RML) paradigm over time as the result of many iterations of the loop of Figure 1.1.

Thinking about the task of intelligence according to Figures 2.2 and 2.3 seems a natural thing to do. The modules seem consistent with the understanding we have of ourselves. They correspond to what we think of when we "look", "think" and "do". The flow of Figure 2.3 seems consistent with how we think we should behave, that we should "look before we leap". It is argued that this model is inappropriate as the final answer, however, for two reasons:

1. It is computationally intractable. Each subsystem must do its task in isolation with little or no focusing knowledge. In the extreme, the perceptual system must interpret the sensors in every possible way to be sure that questions posed by the reasoning system and the motor system will be answered. The reasoning system, in turn, must reason about possibly irrelevant contingencies in the absence of perceptual information. In such a system a considerable amount of reasoning may be duplicated or irrelevant. This situation is intolerable since reasoning tends to be NP complete or worse.
2. The perceptions, reasoning and motor actions done in isolation can easily mismatch each others needs or even be incorrect when viewed in terms of the environment and the goals. The story at the beginning of chapter 1 serves as an illustration. Were the perceptual system of the boatman to interpret the objects as firewood, as did the perceptual system of the savage, the boatman and his friend would never be able to return to the mainland. Reasoning done in isolation and at a macro level are very likely to be incorrect. A detailed plan for a route between two locations, for example, is doomed to be incorrect. It is nearly certain that the environment is not modeled as accurately as would be required to be able to execute a detailed plan without error. It is also unlikely that a motor system can deliver the accuracy required. What results is either incorrect behavior or considerable backtracking, an additional contribution to the intractable character of the system.

The remainder of this section (subsections 2.1.1-2.1.8) describes the major work in the symbolic AI community, showing how each project contributes or relates to the view summarized in Figures 2.1-2.3. Key problems will be highlighted.

2.1.1 Newell and Simon

One of the most influential works in the formulating the symbolic approach to artificial intelligence was that done on human problem solving at Carnegie Mellon University [Newell and Simon, 1972]. A major contribution of this work was formalizing the notion that humans are only one example of a genre called "information" processors. This formalization and the teachings of Newell and Simon are responsible for giving rise to the field of "information processing psychology". Their research in this area, which focused on problem solving, leads to several historically important models of reasoning as implemented in Logic Theorist [Newell, Shaw and Simon, 1963] and the General Problem Solver (GPS) [Newell and Simon, 1963], [Ernst and Newell, 1969] and [Newell and Simon 1972]

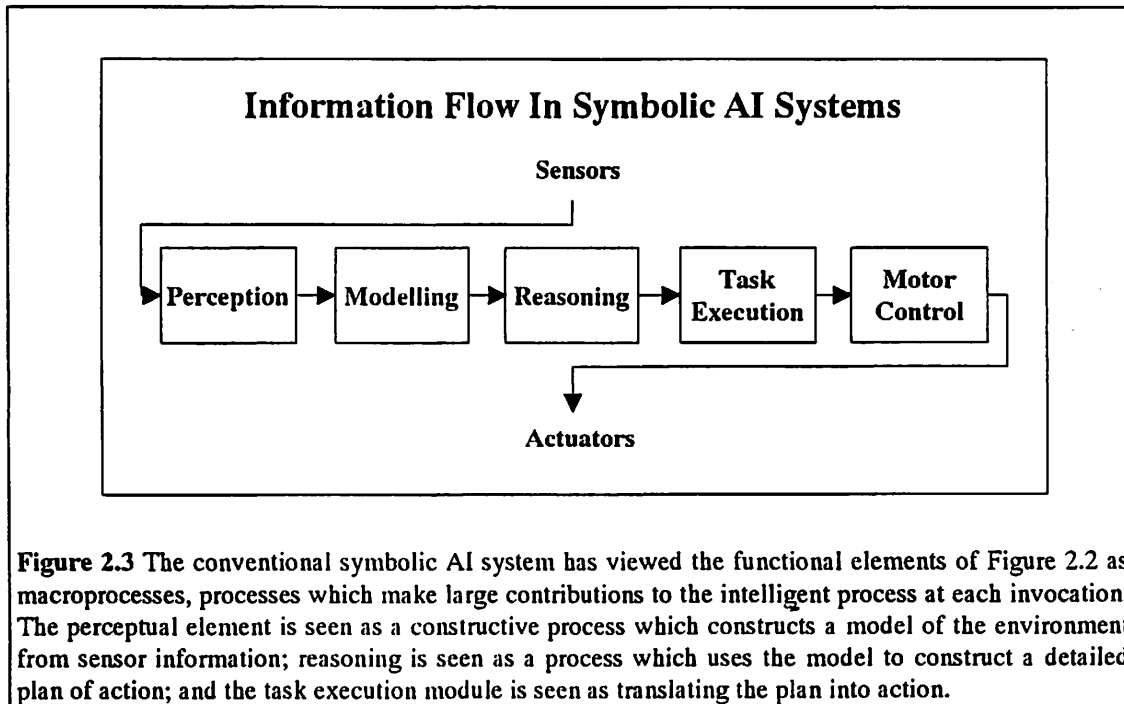


Figure 2.3 The conventional symbolic AI system has viewed the functional elements of Figure 2.2 as macroprocesses, processes which make large contributions to the intelligent process at each invocation. The perceptual element is seen as a constructive process which constructs a model of the environment from sensor information; reasoning is seen as a process which uses the model to construct a detailed plan of action; and the task execution module is seen as translating the plan into action.

The fundamental idea of information processing psychology was that it is *possible* to view an intelligent agent as a system like the one illustrated in Figure 2.1. According to this view the processing element of an "information processing system", as an agent is called, is equipped with a set of instructions referred to as "elementary information processes" (Figure 2.4). Intelligent behavior is viewed as a result of the application of *sequences* (point 5c in Figure 2.4) of these "elementary

information processes" to "symbol structures" in memory (together with those obtained as input from the receptors (Figure 2.4)).

The information processing theory, as summarized in Figure 2.4, did not limit itself to the problem decomposition displayed in Figure 2.2 or to the information flow of Figure 2.3, but Newell and Simon's research in problem solving did. In this research they "... abstracted from the complications of sensory and motor processing; hence we will not treat these aspects in detail."¹ This work did not at all address how perception or action were to be carried out. It began with the assumption that information from the environment was presented to the problem solver as a symbol structure by some other process (e.g. the perception system) and that action details were carried out in response to an output symbol structure by another process, the action subsystem.²

1. There is a set of elements, called "symbols".
2. A "symbol structure" consists of a set of "tokens" (instances) of symbols connected by a set of "relations".
3. "Memory" is a component of an information processing system which is capable of storing and retaining symbol structures.
4. An "information process" is a process that has symbol structures for its inputs or outputs.
5. A "processor" is a component of an information processing system consisting of:
 - (a) a fixed set of "elementary information processes";
 - (b) a "short term memory" that holds the input and output symbols of the elementary information processes;
 - (c) an "interpreter" that determines the sequence of elementary information processes to be executed by the information processing system as a function of the symbol structures in short term memory.
6. A symbol structure "designates" an object if there exist information processes that admit the symbol structure as input and either:
 - (a) affect the object or
 - (b) produce, as output, symbol structures that depend on the object.
7. A symbol structure is a "program" if
 - (a) the object it designates is an information process and
 - (b) the interpreter, if given the program, can execute the designated process.
8. A symbol is "primitive" if its designation is fixed by the elementary information processes or by the external environment of the information processing system.

Figure 2.4. Newell and Simon's definition of an information processing system.

¹ A quotation from page 10 of [Newell and Simon, 1972].

² pp88-89 [Newell and Simon, 1972].

Newell and Simon's work was important in establishing the research methods of both the information psychologist and the symbolic AI community. Equipped with a powerful reasoning capability like GPS it only seems logical to provide it with information from the environment. It is equally natural to propose that this could be done with a perceptual system and that actions could be carried out by a motor system. This is the statement made in Figure 2.2.

2.1.2 The SRI International Robot, "Shakey"

One of the first intelligent robot projects was the SRI International robot "Shakey". This project, described in [Nilsson, 1984], spanned about 6 years, beginning in 1966 and lasting until about 1972. While this is not current research, it is included here because it is a landmark project and it is one of the few robot projects which constructed a complete system and used it to experiment with planning, vision, action. Unlike Newell's effort, which focused on problem solving methods, the goal of the Shakey project was to use problem solving methods (reasoning), together with sensors (perception) and effectors, to solve physical problems. A considerable amount of this research was devoted to understanding how reasoning, perception and action should interact.

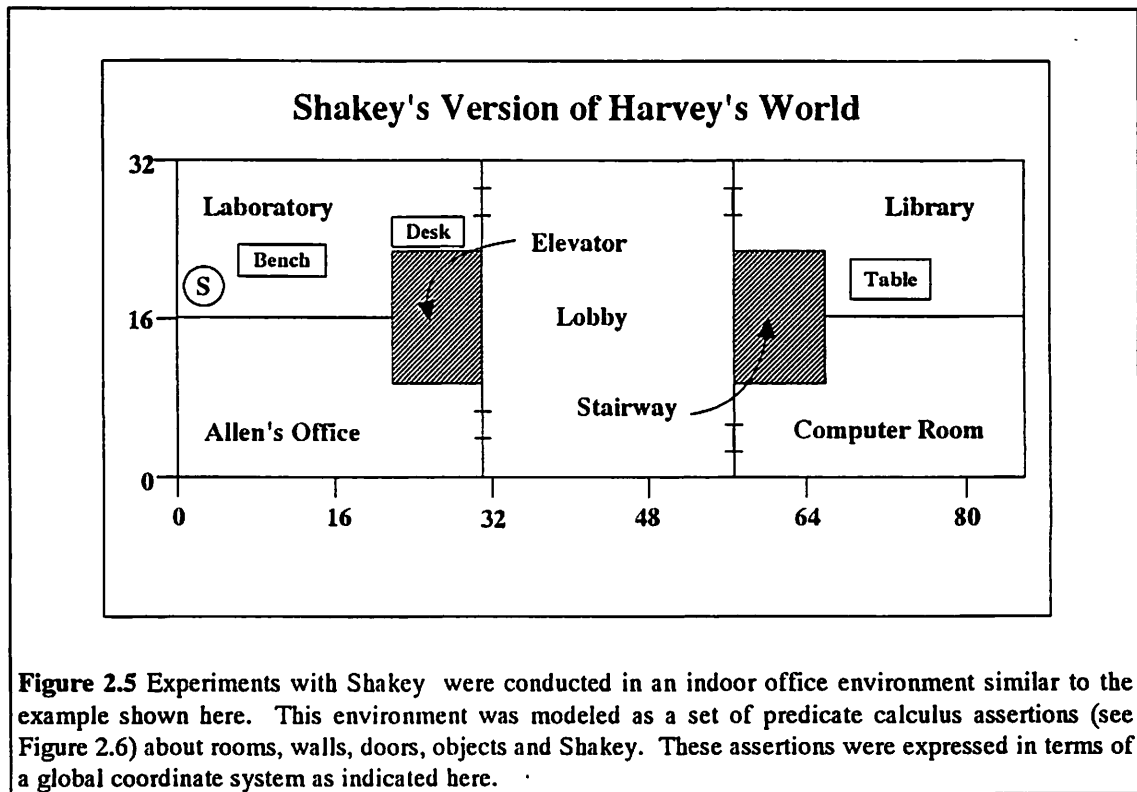


Figure 2.5 Experiments with Shakey were conducted in an indoor office environment similar to the example shown here. This environment was modeled as a set of predicate calculus assertions (see Figure 2.6) about rooms, walls, doors, objects and Shakey. These assertions were expressed in terms of a global coordinate system as indicated here.

Shakey was a mobile vehicle, designed and constructed at SRI. It was capable of moving forward and backward, turning, panning and tilting its "head" and was equipped with a number of sensors. The sensors included a black and white TV camera, an infrared range finder and a number of bump sensors. Processing was done on a PDP-10 which communicated with the vehicle either by radio link or via an umbilical cord. Experiments with Shakey were conducted in a office environment constructed especially for this purpose. Several large boxes were placed in this environment as objects to be used in a variety of tasks designed for Shakey. One of the objectives of the project was to enable Shakey to carry out commands such as:

"Block the door D1 from the room R1 side using Box 1."

To illustrate what this command entails, consider the simplified illustration of Harvey's environment shown in Figure 2.5. At the time the command is given Shakey might be located in the laboratory, as indicated by the circle marked with an "S". In this setting one instance of the command would require that Shakey push library table to a new position which blocks the library door from the library side.

Shakey represented his environment as a collection of predicate calculus assertions similar to those shown in Figure 2.6. Locations mentioned in these assertions refer to a global coordinate system in terms of which all measurements were made. The data for this model was entered by hand, although it was assumed that this information would eventually be derived from sensor data by a perceptual system. The vision system envisioned for this purpose was to be based on region analysis [Brice and Fennema 1970]. Each image was broken into regions of uniform grayscale and then these regions were merged together into larger regions by a pair of heuristics which were based on how the image digitizing process tended to split the image of uniform surfaces into more than one region. The regions which resulted were then collected and grouped together into "objects" using a modification of the procedure developed by Guzman [Guzman, 1968]. Then, based on the shape of the regions and how they were grouped together, the objects in the image were interpreted as floor, wall, baseboard, cube, wedge or unknown. An important modification of this system was later envisioned which interpreted the regions based on a constraint analysis of the allowed relationships between neighboring regions [Duda 1970].

Shakey's Representation of Harvey's World

Rooms

```
type(Laboratory-Main, room)
  roomstatus(known)
  landmarks(Laboratory-Main, (coords (0,16.2,0) (0,32.0, 0) ...))
  boundsroom(Laboratory-Main-Face-1,Laboratory-Main,North)
    faceloc(Laboratory-Main-Face-1, 0.0)
  boundsroom(Laboratory-Main-Face-2,Laboratory-Main,East)
    faceloc(Laboratory-Main-Face-2, 32.0)
  boundsroom(Laboratory-Main-Face-3,Laboratory-Main,South)
    faceloc(Laboratory-Main-Face-3, 21.6)
  boundsroom(Laboratory-Main-Face-4,Laboratory-Main,West)
    faceloc(Laboratory-Main-Face-4, 16.2)
type(Laboratory-Alcove,room)
type (Allen's Office, room)
type (Lobby, room)
type (Library, room)
type (Computer Room, room)
type(Elevator, room)
type(Stairway, room)
```

Doors

```
type(door12, door)
  doorlocs(door12, 23.3, 32.0)
  joinsrooms(door12, Laboratory-Main, Laboratory-Alcove)
  joinsfaces(door12, Laboratory-Main-Face-3, Laboratory-Alcove-Face-1)
  unblocked(door12, Laboratory-Main)
  unblocked(door12, Laboratory-Alcove)
type(door23, door)
```

Objects

```
type(Bench, object)
  at(Bench, 10.2, 21.6)
  dat(Bench, 0.1)
  inroom(Bench, Laboratory-Main)
  radius(Bench, 4.0)
  shape(Bench, box)
  pushable(Bench)
type(Desk,object)
```

Robot

```
type(robot robot)
  at(Harvey, 2.0, 18.0)
  theta(Harvey, 90.0)
  dtheta(Harvey, 0.01)
  dat(Harvey, 0.1, 0.1)
  inroom(Harvey, Laboratory-Main)
```

Figure 2.6 Shakey's world was modeled as a set of predicate calculus statements. The statements in this figure describe the environment pictured in Figure 2.5, an example used throughout this chapter for comparison purposes.

Using this model, tasks like "block the door" were translated into motor commands by a two-tier planning hierarchy. At the top of this hierarchy was STRIPS, a GPS-based hierarchical planner [Fikes and Nilsson, 1971] and [Fikes, Hart and Nilsson, 1971]. STRIPS decomposed each task into a sequence of "intermediate level actions" (ILA's), which were symbolic actions representing complex operations (in many cases these "actions" required more detailed planning). The statement

Go into the next room

would, for example, be converted into the sequence

((goto door-12) (gothru door-12))

of ILA's. This sequence would be passed to another module called PLANEX which took care of more detailed planning. PLANEX would convert (goto door-x) into a sequence of straight line subgoals

(robot-location (x1,y1) (x2,y2) ... (xn,yn) door-location)

When the plan was complete to this level it was converted to low level actions (LLA's): roll and turn.

It was assumed that low level actions were carried out as intended, unless a collision with an obstacle was detected by one of the bump sensors. If no collision was detected a new position and orientation was computed based on dead reckoning. This position was taken as the new robot location and the robot's positional uncertainty was increased. Whenever an LLA terminated due to a collision or if the positional uncertainty of the robot exceeded a threshold, PLANEX invoked the vision system to update the robot position or enter the location of an object. If the preconditions of the next action were still satisfied execution continued; otherwise the remainder of the problem was replanned. The vision system used in the experiments looked for lines in the image to match the model of a known landmark, usually a point on the floor where two walls came together. The position of the image of this point and the orientation of the lines associated with it were used to determine the robot position [Duda 1970].

Planning in this system was a very expensive task. STRIPS used a resolution theorem prover [Green, 1969, 1981] to determine whether or not the preconditions of each ILA operator were satisfied in the model. Resolution is NP-complete in the number of axioms in the model [Lewis and Papadimitriou 1981], so frequent replanning in complex environments cannot be tolerated. As a partial solution to this problem PLANEX learned and used a form of plan schema called a "triangle table". These schemas

offered simplifications to the planning process by providing prepackaged parameterized plans, making replanning less often necessary. As the system solved more problems, its planning became more efficient.

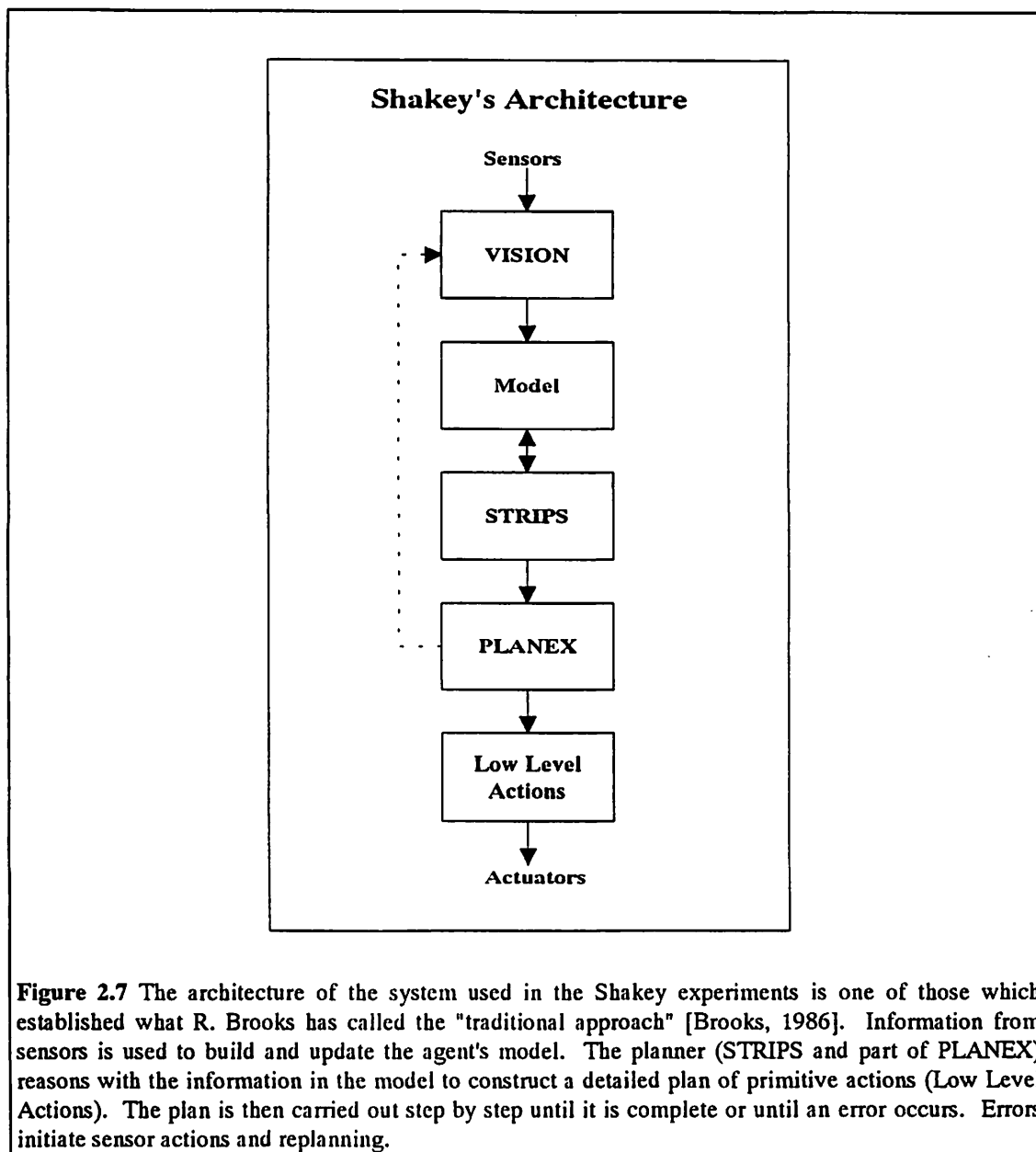
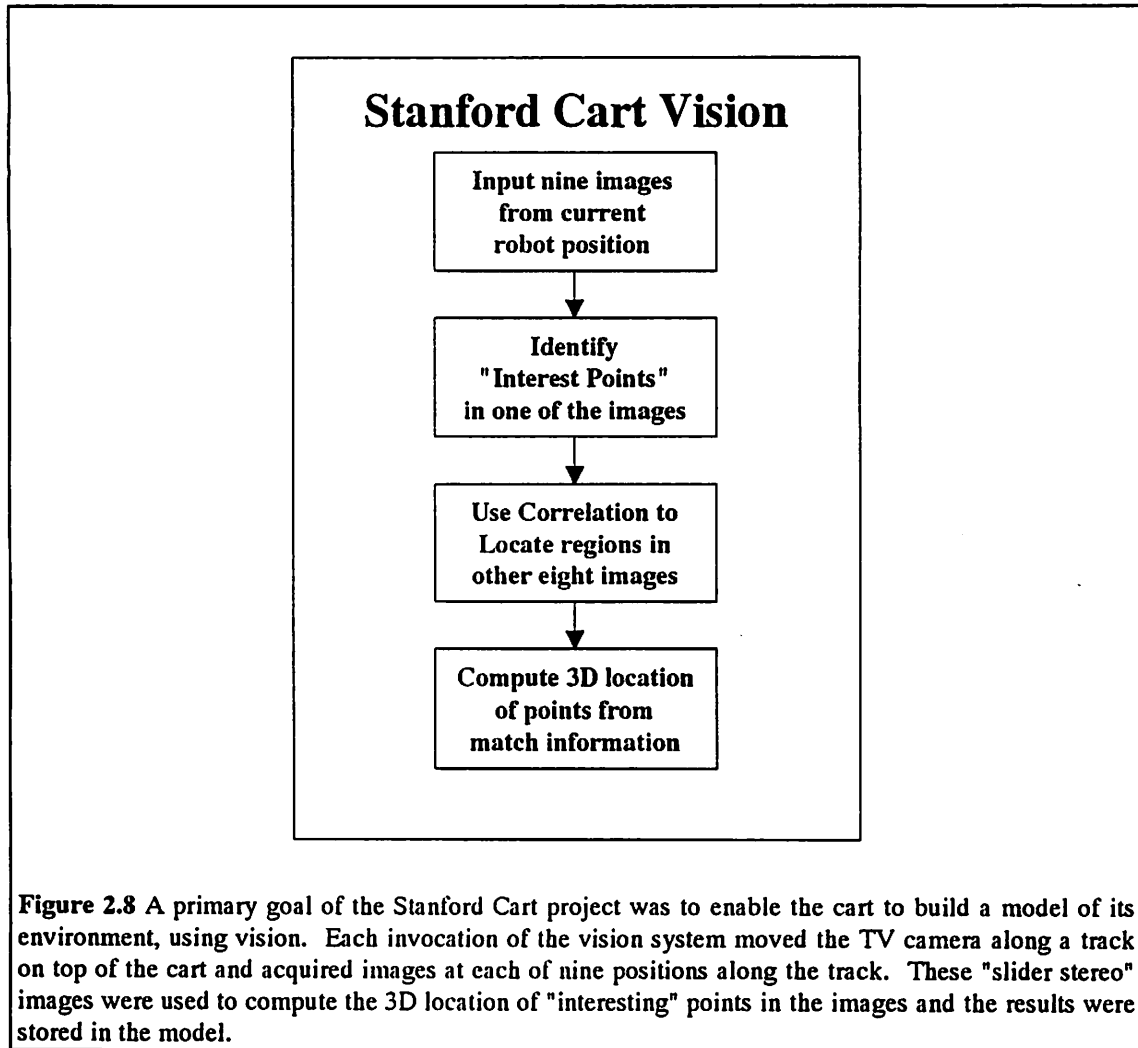


Figure 2.7 The architecture of the system used in the Shakey experiments is one of those which established what R. Brooks has called the "traditional approach" [Brooks, 1986]. Information from sensors is used to build and update the agent's model. The planner (STRIPS and part of PLANEX) reasons with the information in the model to construct a detailed plan of primitive actions (Low Level Actions). The plan is then carried out step by step until it is complete or until an error occurs. Errors initiate sensor actions and replanning.

The overall architecture used in the experiments with Shakey is summarized in Figure 2.7. As described in the preceding paragraphs, Shakey's intellectual system was divided into three macroprocesses: VISION, which interprets sensory information and enters it into the model, STRIPS,

which produces a complete and detailed plan in terms of ILA's, and PLANEX, which carries out the detailed plan of ILA's. This architecture divides the intelligent process into the functional elements indicated in Figure 2.2 and follows the information flow illustrated in Figure 2.3. This system was not robust, but it did show that a number of rather complex problems could be solved in controlled environments.



The primary difficulty with the solution was that it did not scale up to more complex environments. One of the major factors contributing to this was due to the complexity characteristics inherent in the STRIPS reasoning process, which ultimately depended on a resolution theorem prover. The system architecture made matters worse. Whenever PLANEX discovered an error in execution it would restart

the process by invoking the VISION module to update the model. STRIPS would then be recalled to develop a new plan from the current location.

2.1.3 Stanford Cart

The Stanford cart [Moravec 1983], another historically important robot project, focused on using vision to navigate in largely unknown environments. Experiments with the cart were performed in both indoor and outdoor environments. Beginning with no knowledge of the specific environment, the goal was to be able to move from the starting location to a new, specified location without hitting any objects.

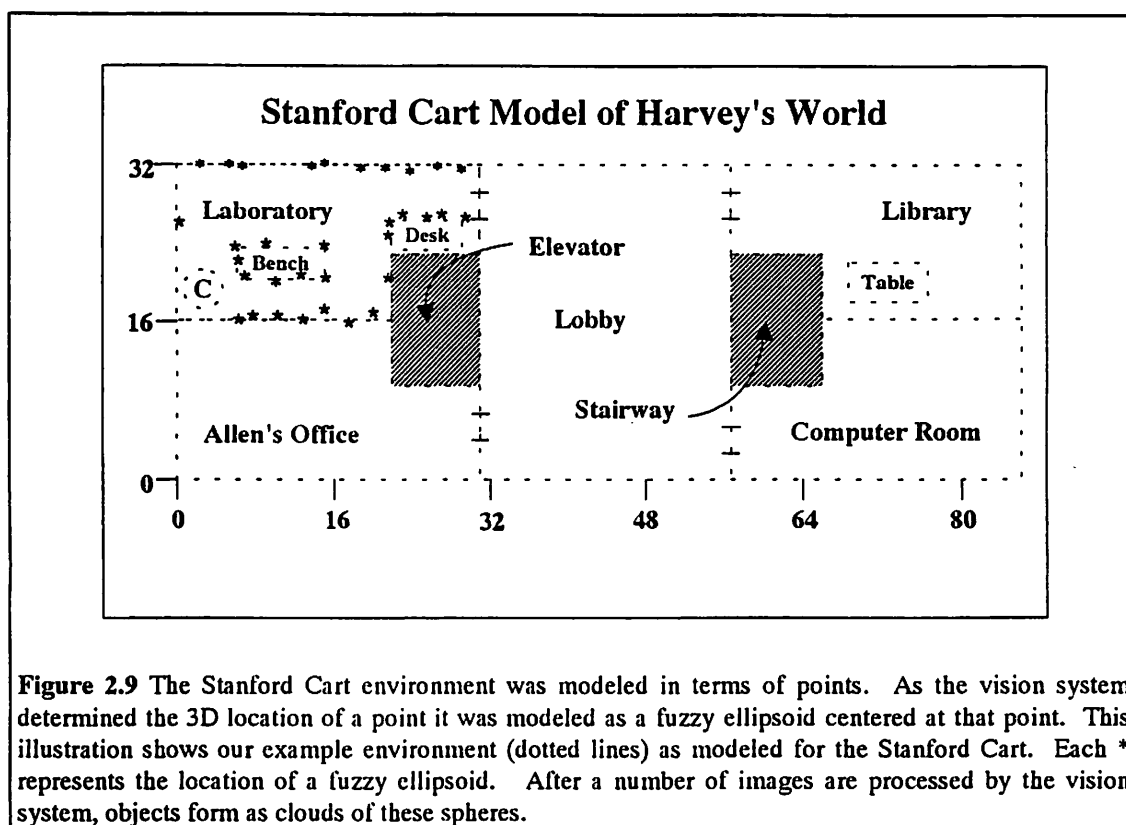


Figure 2.9 The Stanford Cart environment was modeled in terms of points. As the vision system determined the 3D location of a point it was modeled as a fuzzy ellipsoid centered at that point. This illustration shows our example environment (dotted lines) as modeled for the Stanford Cart. Each * represents the location of a fuzzy ellipsoid. After a number of images are processed by the vision system, objects form as clouds of these spheres.

The cycle of operation of this system began with the vision system. This system gathered 3D information using what Moravec called "slider stereo". The cart was equipped with a mechanism which would slide the camera to nine predetermined positions along a straight line. One "look" analyzed the data from all nine images. The vision process (Figure 2.8) identified between 30 and 40 points of "interest" in the center image, using what has since become well known as the "Moravec interest operator" [Ballard and Brown, 1982]. Then, using correlation, the matching point for each of these

"interest points" was located in the other 8 images. Knowledge of the camera position, the location of the points in the image and their offsets in the various images were used to determine the 3D location of each of the interest points. These 3D locations were entered into a model.

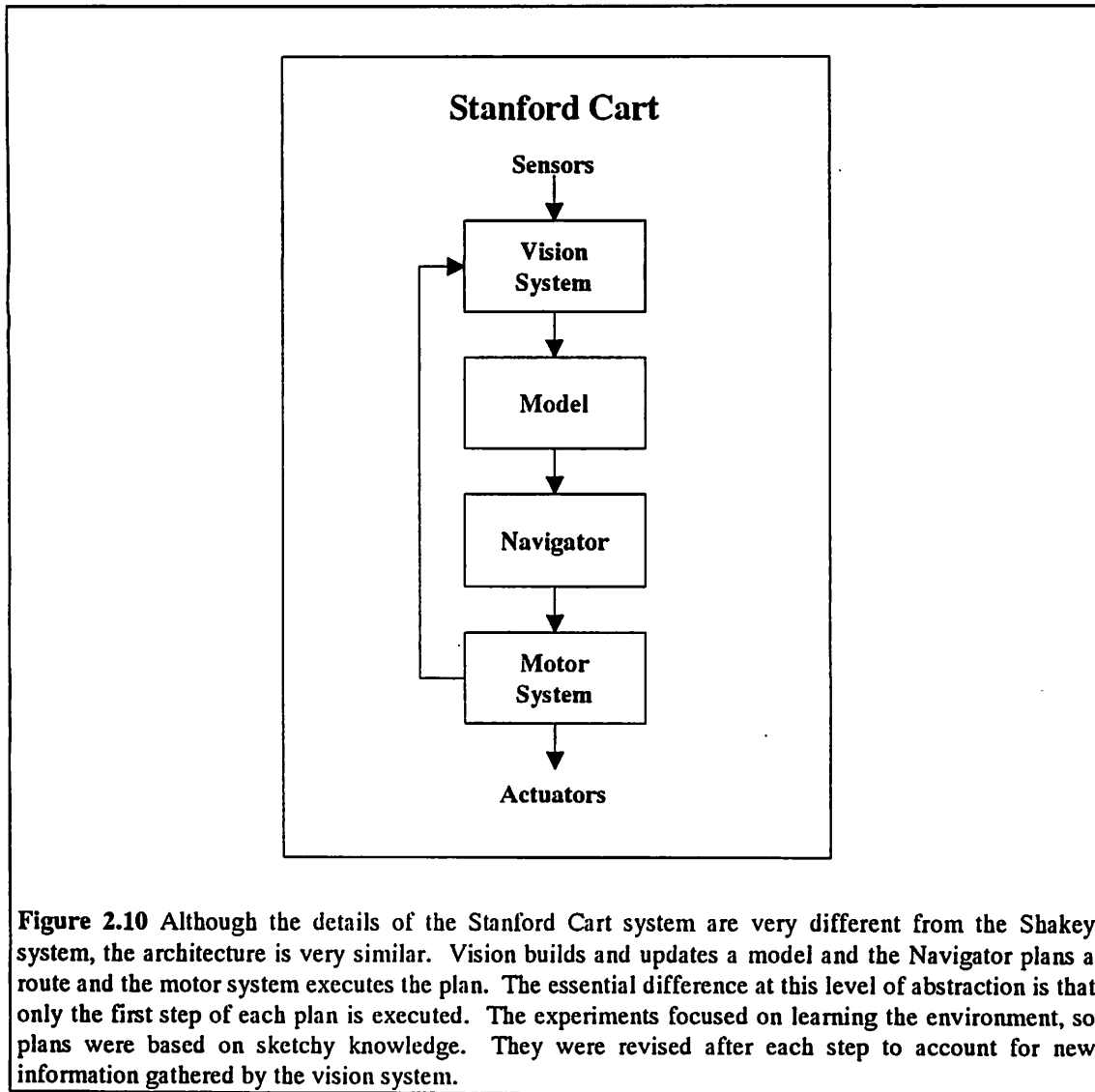
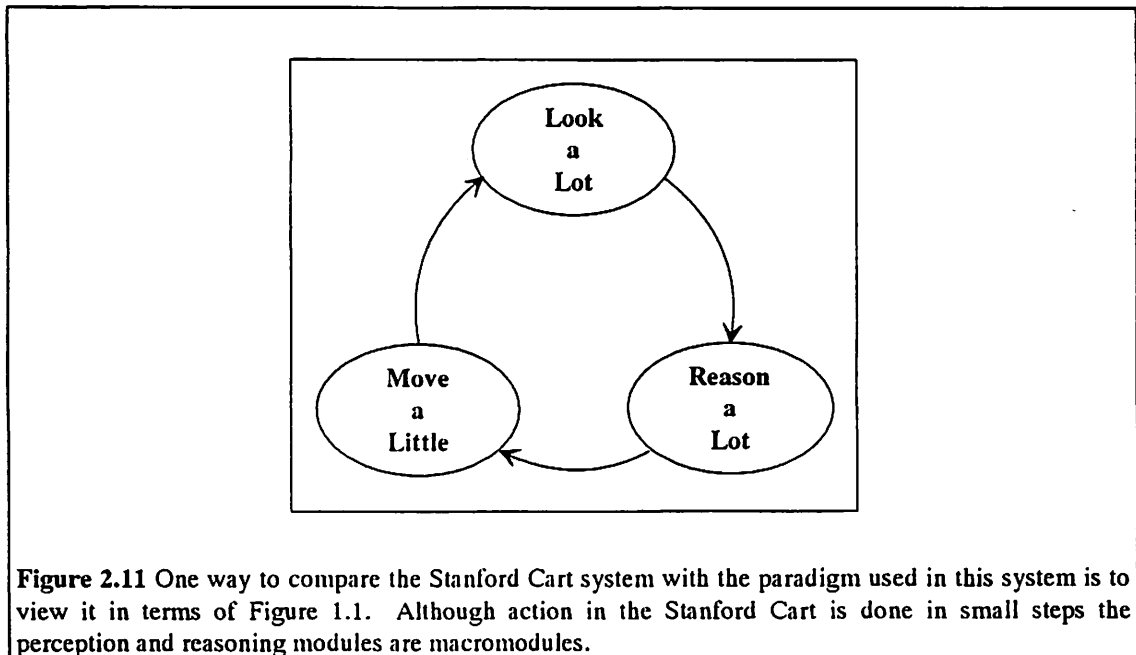


Figure 2.10 Although the details of the Stanford Cart system are very different from the Shakey system, the architecture is very similar. Vision builds and updates a model and the Navigator plans a route and the motor system executes the plan. The essential difference at this level of abstraction is that only the first step of each plan is executed. The experiments focused on learning the environment, so plans were based on sketchy knowledge. They were revised after each step to account for new information gathered by the vision system.

The model was based on a single global 3D coordinate system (Figure 2.9). Each location identified by the vision system was entered as a sphere centered at that location with a radius representing the uncertainty in its location. Objects and walls developed over a number of "looks" as clusters of intersecting spheres. The robot itself was modeled as a disk 3 meters in diameter.

Once the vision information had been entered into the model a plan was constructed. Spheres more than a certain distance above the floor were considered not to be potential obstacles. These spheres were ignored and the remaining spheres were projected as circles onto the floor plane. The diameter of these circles were then "grown" by the diameter of the robot. The planning procedure began by finding all the straight line segments which were tangent at their endpoints to some pair of these circles. The path was constructed using a shortest path graph search from these line segments and the circular arcs connecting them. This computation was $O(n^3)$ in the number n of 3D points in the model.

When the plan was complete the cart would move a short distance (less than 1 meter) in the direction of the first planned segment. This motion was carried out in a dead reckoning fashion. If the goal had not yet been reached when the motion was complete, the system would then reinvoked the vision system and the cycle would begin again.



While the details of this system were very different from the SRI system, the basic architecture (Figure 2.10) is similar. The system is constructed of the macroprocesses of Figure 2.2 and these macroprocesses observe the information flow of Figure 2.3. As with Shakey, the vision system directly entered information into the model. The resulting model was used by a planner to generate a detailed plan and the motor system carried out the plan. An essential difference between the Stanford Cart and

SRI's Shakey is that vision was used after every step to further develop the model and plans were constructed anew after each step to incorporate new information.

Figure 2.11 illustrates the Stanford system in terms of the paradigm introduced in this dissertation. One difference is in the granularity of the information contributed by each process invocation. The other is that the vision process is autonomous. It is not at all affected or focused by the reasoning process.

"One of the most serious limitations [of the system] was the excruciating slowness of the program. In spite of my best efforts and many compromises in the interest of speed it took 10 to 15 minutes to acquire and consider the images at each meter long lurch, on a lightly loaded DEC KL-10. This translated to an effective Cart velocity of 3 to 5 *m[eters] an hour*."³ The rather complete vision analysis at each step, using correlation to match 30-40 points in 9 different images, was a main contributor to the computation time. It is clear, however, that the complexity of the planning algorithm ($O(n^3)$) also becomes quickly a problem for large spaces, especially since n refers to the number of *features* detected by the vision system.

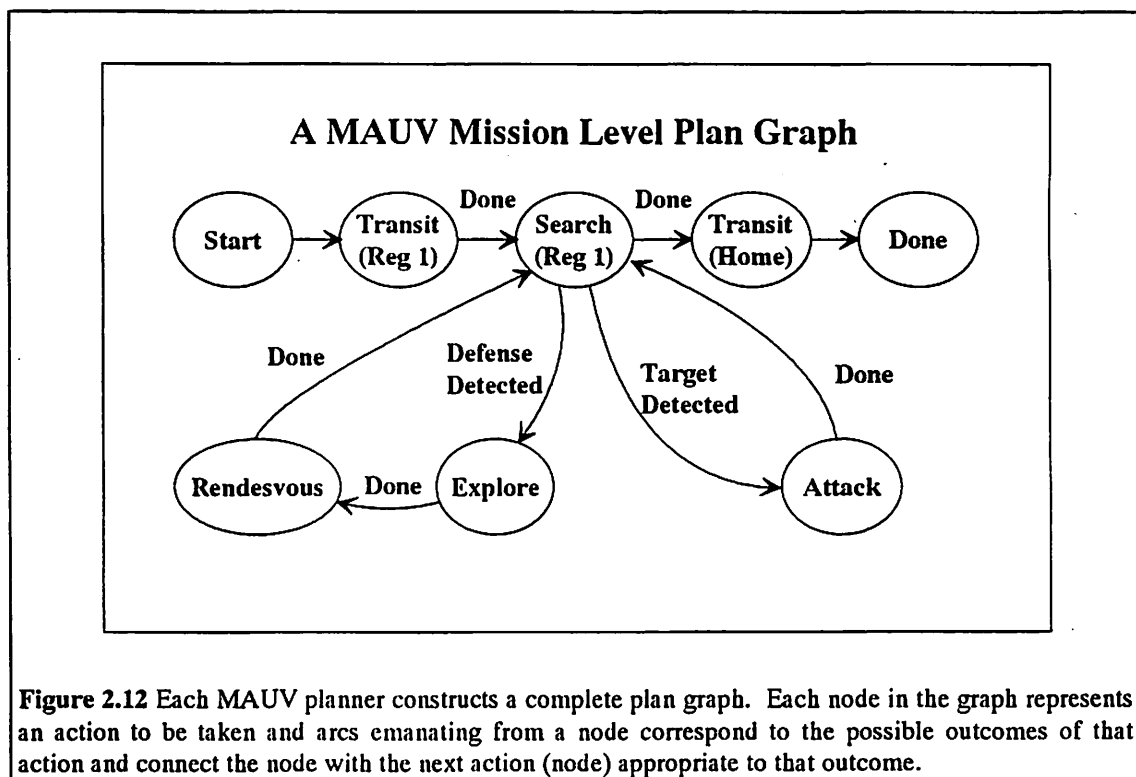
2.1.4 The Multiple Autonomous Undersea Vehicles (MAUV) project

The National Institute of Standards and Technology project "MAUV" had the goal of designing a system which would be able to control of multiple vehicles in the real time execution of cooperative underwater tasks [Herman and Albus, 1987, 1988, 1988-89], [Herman, Hong Swets and Oskard 1988] and [Herman, Albus and Hong 1990]. In this problem it was conceived that a number of agents specially designed for underwater tasks would work together under the direction of a central control system to perform tasks such as mapping the bottom of underwater areas [Oskard and Hong 1988] or searching for underwater targets.

This system as a whole was designed to have an elaborate hierarchical control structure which decomposed reasoning (planning) activities into several levels of abstraction: the mission level, the group level, the vehicle level and e-move levels. The planning effort was distributed amongst the vehicles according to this abstraction hierarchy. Each planner would convert tasks received from a higher level

³ page 877 of [Moravec 1983].

planner into a complete plan graph (Figure 2.12). The nodes in this graph represent tasks to be performed by the next lower level and the arcs emanating from a node correspond to all the possible outcomes of that task. Each arc connects a node with the node representing the next action to perform, given that the outcome corresponding to the arc is true. The search node in Figure 2.12, for example, has three possible outcomes: a target was detected, defence was detected, or the search was completed. Planning began at the mission level. As each level completed its plan it began executing it step by step, invoking lower level agents to carry out each task. Thus, the mission level creates a plan graph and invokes group level agents. The group level agents, in turn, would create plan graphs and invoke vehicle level agents. The vehicle level agents create their plan graphs and invoke e-move agents which, in this case, carry out primitive steps. Creating complete plan graphs is a very costly computation but planning in real time was possible in this system because this cost was distributed over many agents at various levels.



This system as a whole is reminiscent of Minsky's Society of Mind [Minsky, 1986]. The vehicles would act together as society of agents, accomplishing tasks in real time by distributing the work and organizing the results. This aspect of the MAUV project has exciting, but as yet unrealized, potential. It

will be discussed further, along with some of Minsky's ideas, in section 2.2.1. This section will focus on a discussion of the individual agents.

The architecture of these individual agents is made up of three processes: a planner, a sensory processing unit and an executor. At the vehicle level sensory processing is done on raw sensor data; the planner builds a plan graph consisting of actions which correspond to vehicle motions; and the executor converts the plan into actual motions. Higher level agents work with abstractions of these processes. Sensory processing in these higher level agents is done on information which has been partially processed by the agents it controls; planning produces plan graphs of commands to be implemented by lower agents; the executor issues these commands to those agents.

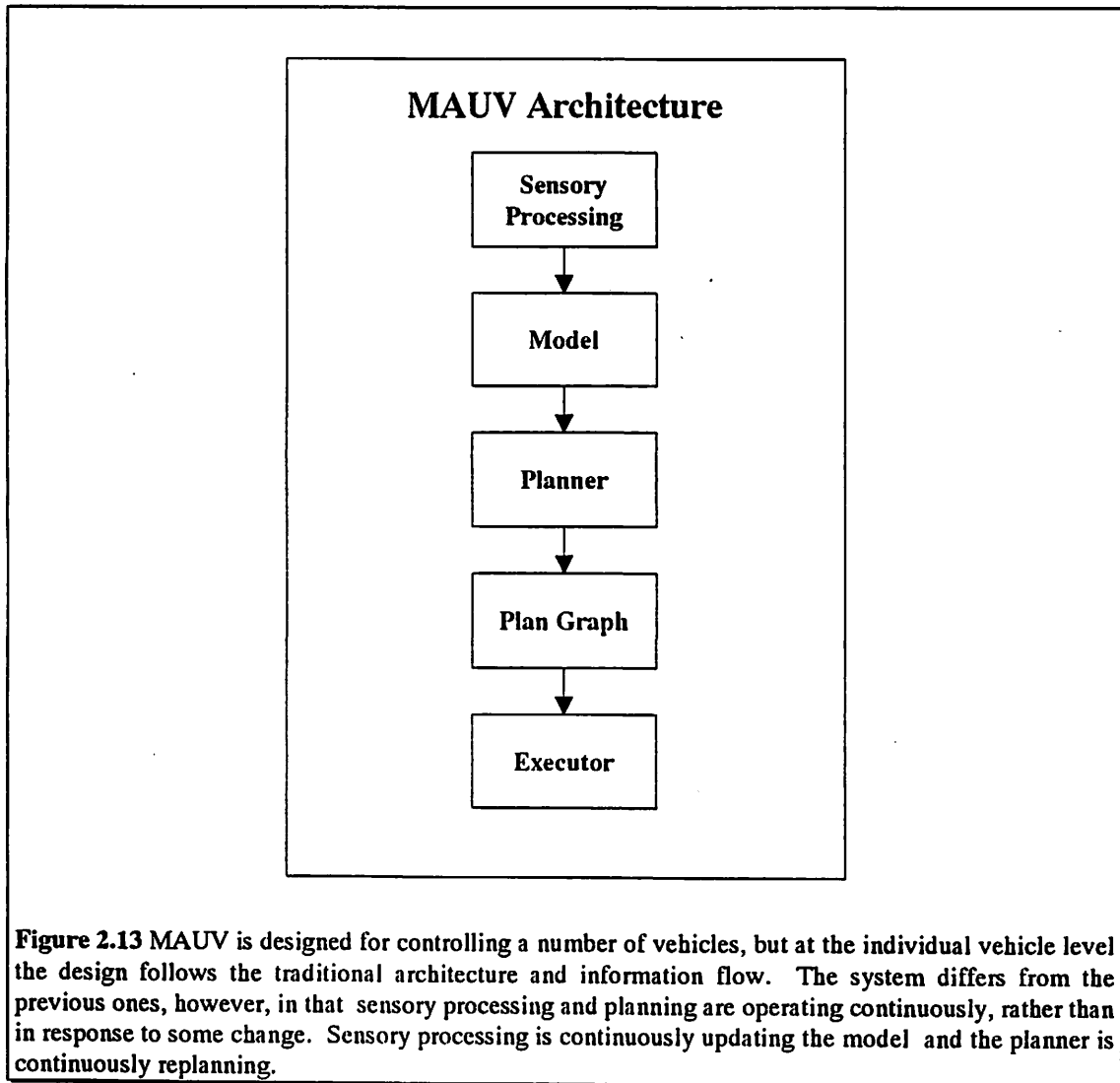


Figure 2.13 MAUV is designed for controlling a number of vehicles, but at the individual vehicle level the design follows the traditional architecture and information flow. The system differs from the previous ones, however, in that sensory processing and planning are operating continuously, rather than in response to some change. Sensory processing is continuously updating the model and the planner is continuously replanning.

As with other systems described so far, the sensory processing unit for each agent was a macroprocess. This unit fuses sensory input data from a variety of sensors and performs spatial and temporal integration in such a manner as to detect events and recognize features, objects and relationships in the world. The results of this analysis is stored in the world model which acts as a buffer between the sensory process and the other two processes [Herman, Albus and Hong, 1990].

Planning in each agent was also a macroprocess. Each invocation constructed a complete plan graph based on the current command and what was stored in the model. Information about the world required for planning and execution were obtained from this world model. The planning process was repeated at regular time intervals so that the system could react to new sensory information as it is integrated into the model.

Execution was also a macroprocess which carried out complete commands as determined by the current plan graph and information in the model. The results of the sensory processing stored in the model are used by the agent to determine which arc should be traversed in its planning graph. In Figure 2.12, for example, if a target was detected during the search action then the agent would perform (or issue) the attack command.

In summary, the architecture of each agent was composed of three macroprocesses: sensory processing, planning and execution, as in Figure 2.13. Each agent in this system has the architecture illustrated in Figure 2.2 and the flow of information illustrated in Figure 2.3. Viewed as a society, the ideas of this research might be modified to describe a single agent. As was pointed out earlier in this subsection, this design would look similar to some of the connectionist work, in particular the work of Minsky [Minsky, 1986]. MAUV will be revisited in that context in section 2.2.1.

2.1.5 The NAVLAB project at Carnegie-Mellon

The Carnegie Mellon "NAVLAB" project [Thorpe, Herbert, Kanade and Shafer, 1988] has the goal of developing perception and navigation techniques for controlling a mobile robot. NAVLAB, the robot vehicle, is a commercial van furnished with onboard computers, rigged to control the van via servos. The majority of the work in this project has been directed towards developing the perception capabilities

necessary for steering NAVLAB down a road, but the project has proposed an overall system design of which this steering capability is a part.

The system architecture takes the form of a modified blackboard (Figure 2.14). Each of the shaded boxes in the figure represents a program which runs independently of the others, except to the extent that it communicates with the others via the central database (the blackboard). The map navigator maintains a global map, incorporating new information as it becomes available from perceptual activities, and plans global paths. The pilot performs three essential functions: it uses the map and knowledge of the NAVLAB position to predict the road location as an input to the vision module; it plans local polyline paths which avoid known obstacles; and it keeps the vehicle on the road. The Helm module converts the path generated by the Pilot into smooth arcs and steers the vehicle as dictated by these arcs. Perception is implemented as two modules: an obstacle detection module, which returns a list of obstacles near the current position of the NAVLAB, and a road position (Color Vision) module, which returns the position of the next section of road. Both vision modules operate in response to pilot requests.

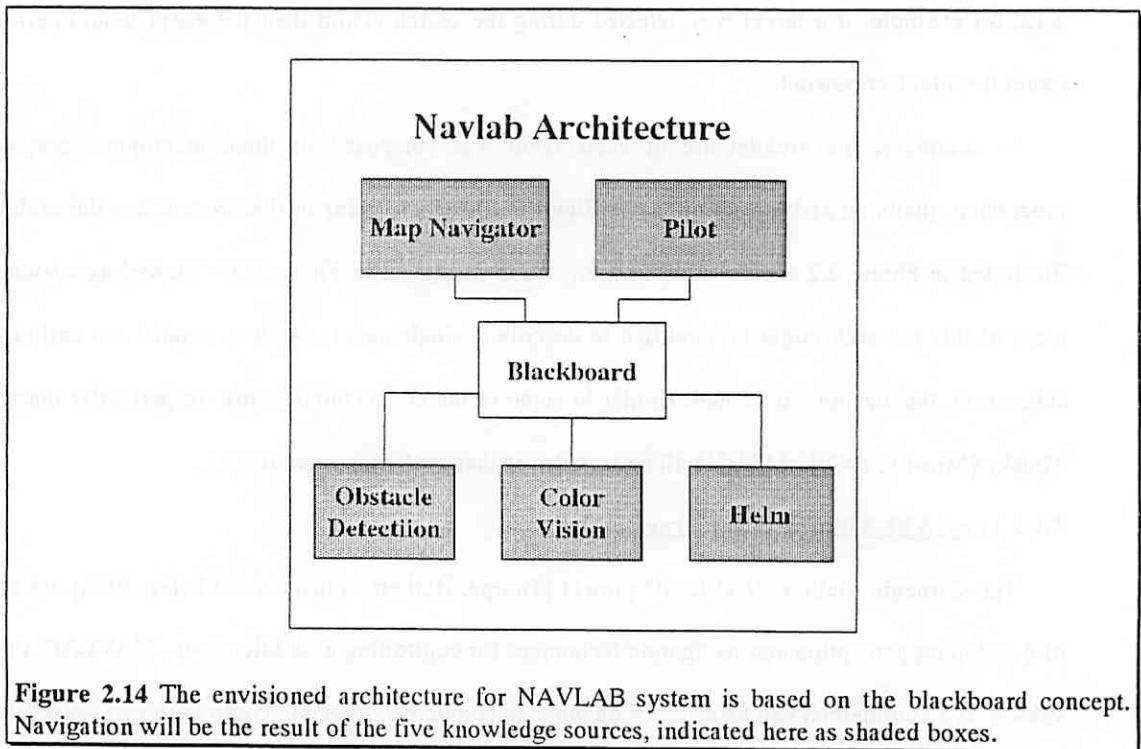


Figure 2.14 The envisioned architecture for NAVLAB system is based on the blackboard concept. Navigation will be the result of the five knowledge sources, indicated here as shaded boxes.

The environmental model is somewhat distributed. The blackboard contains information about roads, junctions and the position of the NAVLAB, but the color vision module contains and maintains a model of road appearance in terms of its color, texture and local shape characteristics. The color vision module models the environment as consisting of only two categories: road and non-road. To cope with the color variations of these environmental categories they are each represented by a number of color classes (four were used in the reference cited). The color properties of these classes are represented by their mean red (R), green (G) and blue (B) values, a 3x3 RGB covariance matrix and the fraction of pixels expected a priori to be in that class. Texture is modeled at each point in terms of the number of pixels with high gradient value in an area surrounding that point. The Roberts-cross gradient is calculated in a fixed area around the point at high resolution (high-frequency) and at low resolution (low-frequency). These gradients are then thresholded and the number of pixels exceeding threshold in each case is determined. These counts are given the names high-frequency-gradient and low-frequency-gradient, respectively. Texture at a point is then given the value:

$$\text{texture} = \frac{\text{high-frequency-gradient}}{\alpha * \text{low-frequency-gradient} + \beta * \text{mean-pixel-value}}$$

The values of α and β are determined experimentally. Road shape is modeled by declaring that the road is locally flat, is of constant width and locally straight.

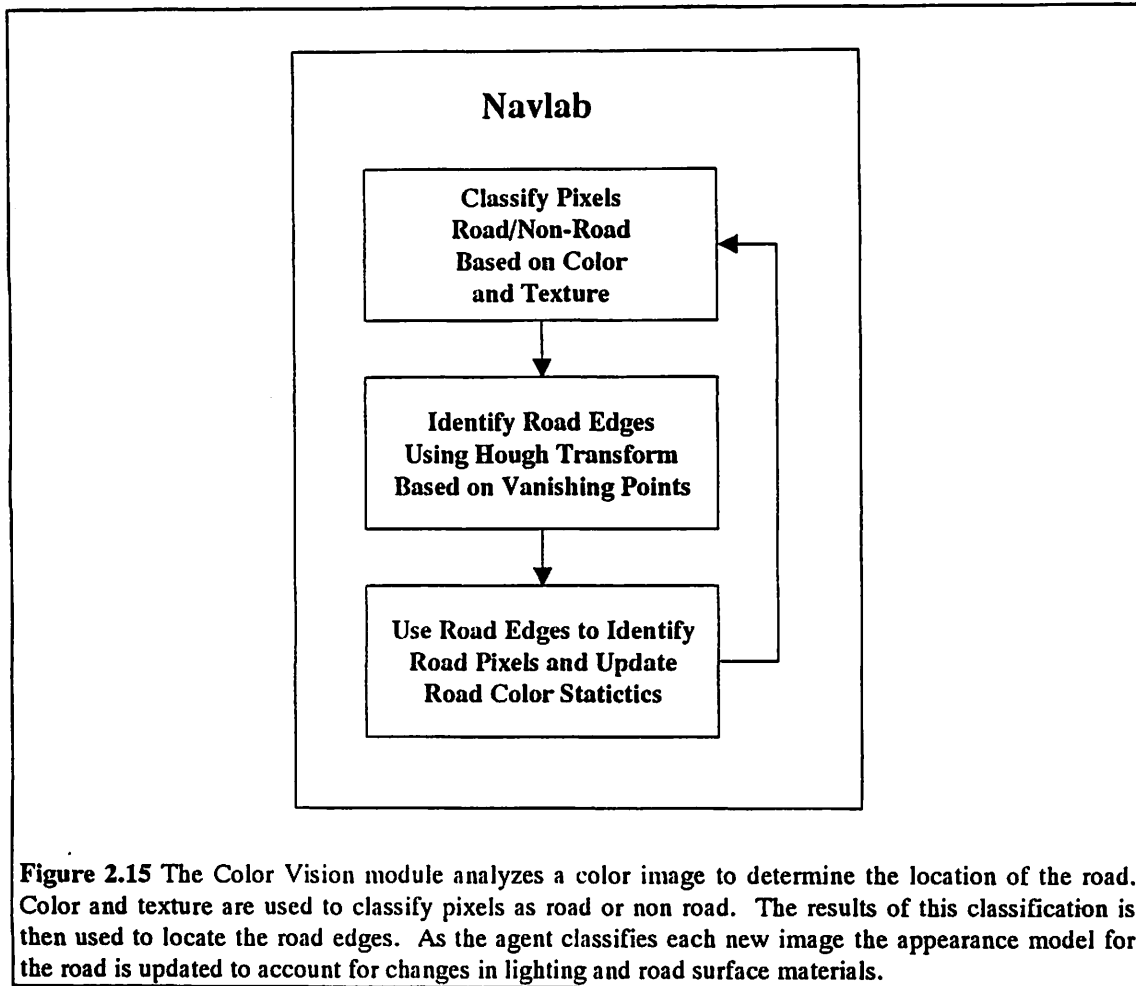
Using the appearance model, the color vision module finds the position of the road in the image following the outline of Figure 2.15. Each pixel is classified as road or non-road based on color using a maximum likelihood classifier and on texture based on how its texture value compares with the values associated with the road and non-road classes. Associated with each of these classifications is a confidence measure. These results are then combined for each road and non-road class using the formula: for each road and non-road class i

$$\text{confidence}(i) = (1-\alpha) * \text{confidence-texture}(i) + \alpha * \text{confidence-color}(i).$$

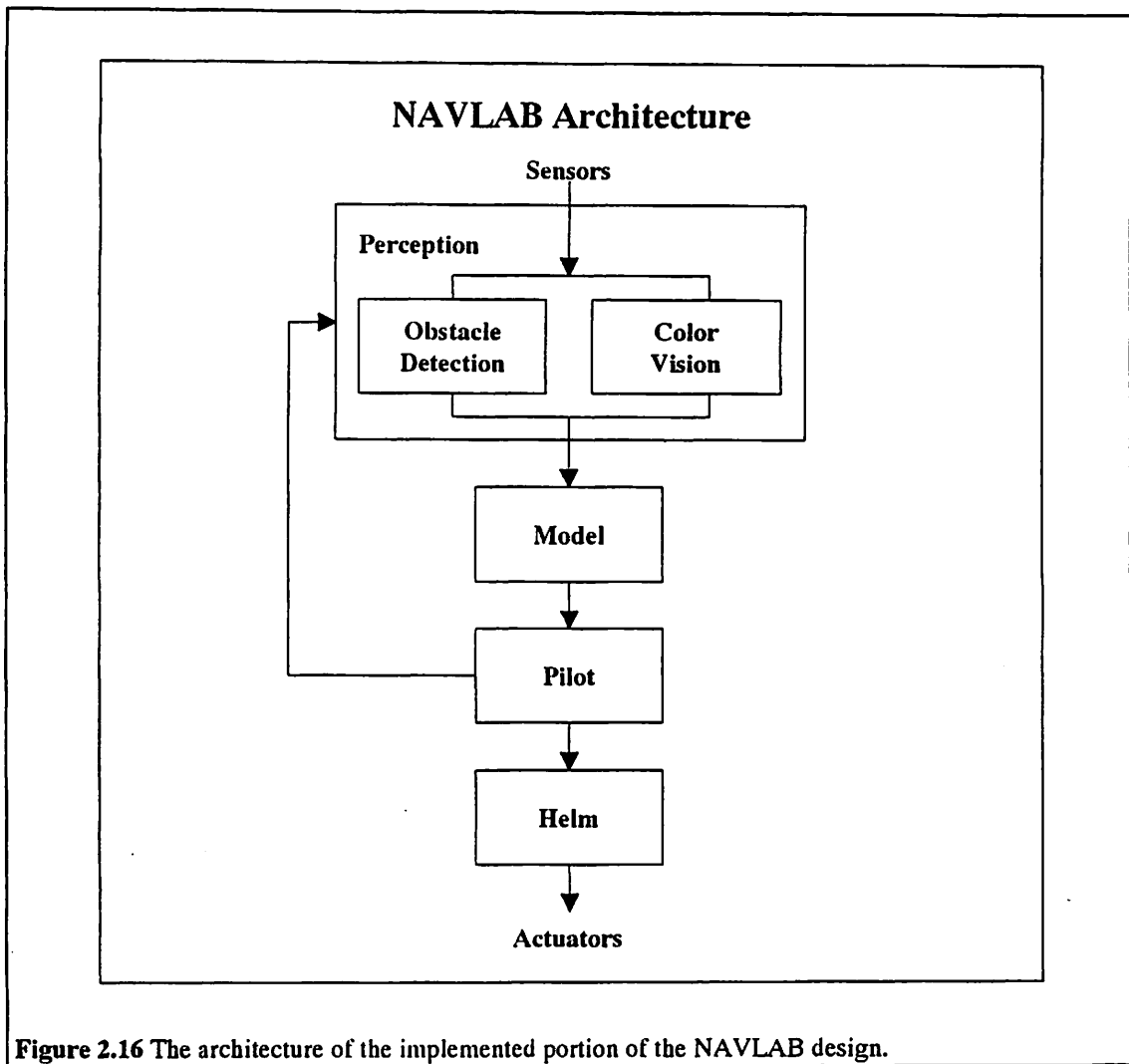
These confidences are then combined into an overall-confidence number:

$$\text{overall-conf} = \text{Max}\{\text{confidence}(i): i \text{ is a road class}\} - \text{Max}\{\text{confidence}(i): i \text{ is a non-road class}\}.$$

Pixels with positive overall-confidence are classified as road. All other pixels are classified as non-road.



Once all the pixels have been classified, the "most likely" position of the road in the image is determined. Since the road is assumed to be locally flat and straight, it can be assumed that the edges of the road can be approximated by straight lines which will meet at some vanishing point on the horizon. Using knowledge of the road width, the family of possible road descriptions can be parameterized by the position of the vanishing point P and the angle Θ made by the centerline of the road with the horizon. Using this, each pixel votes for the combinations of P and Θ consistent with its interpretation: pixels classified as road vote positively if it would be part of the road specified; pixels classified as non-road vote negatively for combinations consistent with it being on the road. The pair which receives the most votes is stored in the blackboard model as a description of the next section of road and the pixels consistent with this road description are used to update the class color and parameters to account for changes in the road surface and illumination over time.



The implemented portion of the proposed system of Figure 2.14 is shown in Figure 2.16. It is assumed that the Map navigator has planned a global path which provides global direction to the pilot. The responsibility of the system in Figure 2.16 to carry out this plan. This is accomplished by first invoking the perceptual process which constructs a model of the local environment. This perceptual process is implemented as two macroprocesses which operate in parallel. The Color vision module, described above, analyzes the image to get a detailed model of the next section of road. The obstacle detection macromodule analyzes range images to determine the nature of the terrain and locate obstacles. The results of these two analyses are stored in the model. Then, the pilot constructs a detailed plan for the next leg of the global plan using this new local model. This detailed plan is constructed as a polyline

path which avoids obstacles and stays on the road. Finally the Helm module smooths this polyline path and sends signals to the servos which are intended to move the vehicle along this path. This sequence repeats until the goal is achieved.

NAVLAB is another example of the processes division illustrated in Figure 2.2 and, although NAVLAB is implemented using the blackboard architecture, the flow of information still follows the diagram of Figure 2.3.

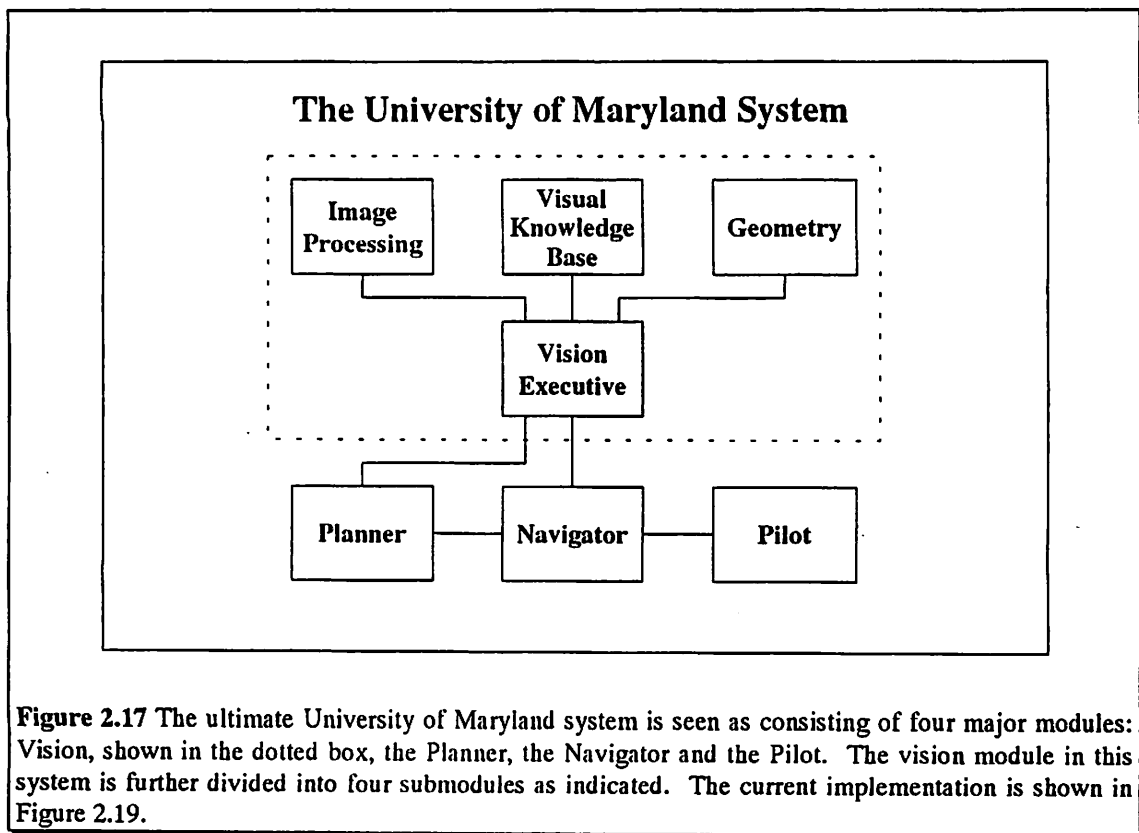


Figure 2.17 The ultimate University of Maryland system is seen as consisting of four major modules: Vision, shown in the dotted box, the Planner, the Navigator and the Pilot. The vision module in this system is further divided into four submodules as indicated. The current implementation is shown in Figure 2.19.

2.1.6 The University of Maryland System

The Autonomous Land Vehicle project at the University of Maryland has the goal of designing a system to do autonomous navigation using vision [Waxman, LeMoigne, Davis, et al. 1987]. The agent used in this program consists of a black and white CCD camera moved about through a model environment by a robot arm. The environment used in the experiments is a large table-sized artificial terrain consisting a network of dark strips, representing roads, laid out over model hills, valleys and flat areas. The robot arm moves the camera through this environment as if it were traveling on the surface, controlling the

camera height and orientation using information obtained from linear position sensors which are in contact with the model surface. The majority of the research effort in this project has been directed at perception but some thought has been given to how this perceptual system will fit into a final system.

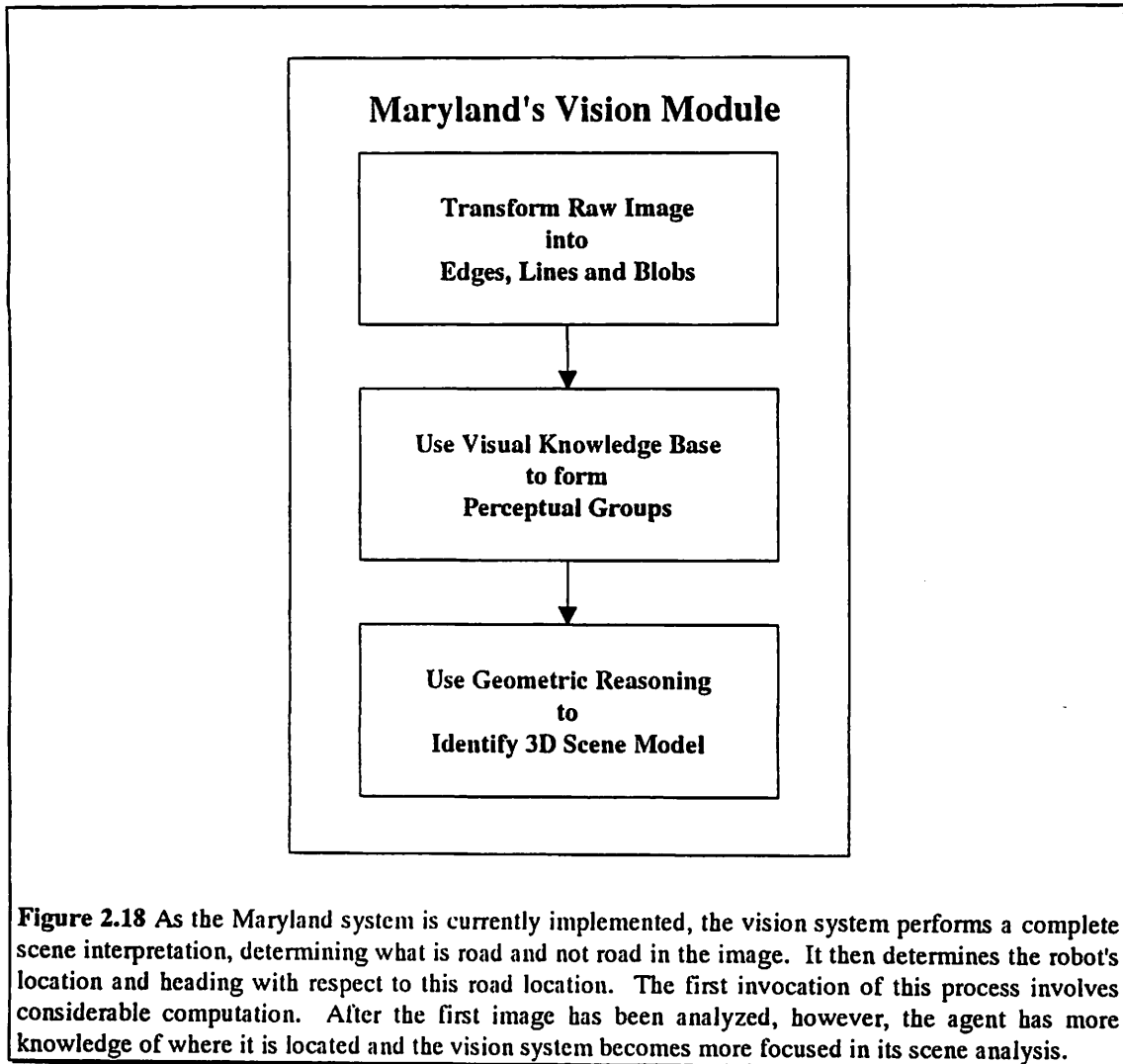
The Maryland system design (Figure 2.17) breaks the navigation process into four submodules: a planner, a navigator, a pilot and a vision system. The vision system is further broken down into a vision executive, an image processing module, a visual knowledge base and a geometry module. In this system, the planner is responsible for establishing overall goals of the agent. The responsibility of the Navigator module is to determine the location of the agent and to plan a path through the environment to the goal. Path planning is done at three levels of abstraction: long, intermediate and short range.

Long range path planning generates a sequence "regions", the first of which contains the location of the agent and the last of which contains the goal. Each of the regions in the sequence would correspond to an area of the environment which is homogeneous with respect to some property, such as uniform visibility of landmarks or ease of navigability. Consecutive regions in the sequence would be spatially adjacent. A hallway would be an example of such a region in an indoor environment. Typical regions in an outdoor environment might be a forest or a plain.

Intermediate range path planning selects a "corridor" of free space through which the vehicle will next travel. This corridor should be free of known obstacles and should lead towards the next region in the sequence produced by the long range navigator. Vision is used to classify the area into free space and non-free space and to update the model before the corridor selection process begins. Indoor corridors in an uncluttered region would usually be the same as the region itself. A corridor in a cluttered region would be a sufficiently wide path through the clutter. In an outdoor region a corridor would correspond to a road or path. Driving a car along a road is seen as an example of this kind of navigating.

Short range path planning constructs a detailed path through the established free space corridor. It is at this level that obstacle avoidance is considered. Vision plays a role in this portion of the system. When a short range path plan has been constructed it is executed. Vision is used to keep the vehicle on track, to detect obstacles and to update a local map of the environment. Planning is based on this local map.

Execution of a short range path plan is carried out by the Pilot. The Pilot converts this planned path into motion and informs the navigator of its progress. No perception is used by this module. It measures progress based on readings from effector shaft encoders (dead reckoning).



The general role for vision as stated by the authors is: "The Vision system as a whole is responsible for perceiving objects of interest (e.g. roads and landmarks) and representing them in an 'object centered' reference frame."⁴ The system designed to accomplish this consists of an elaborate reasoning and processing system made up of the four modules identified in Figure 2.14. The vision executive coordinates the flow of processing between the three other elements: the image processing module, which

⁴ page 128, Waxman, LeMoigne, Davis, et al. 1987

performs low level vision operations; the visual knowledge base, which performs intermediate and high level vision processing; and the geometry module, which produces three-dimensional models.

The environmental model of this system has not been described in detail but it was suggested that it would be distributed. The appearance-related features of the environment, including objects and known landmarks, would be resident in the vision system whereas topography data, along with the location of known landmarks, would be associated with the navigator module.

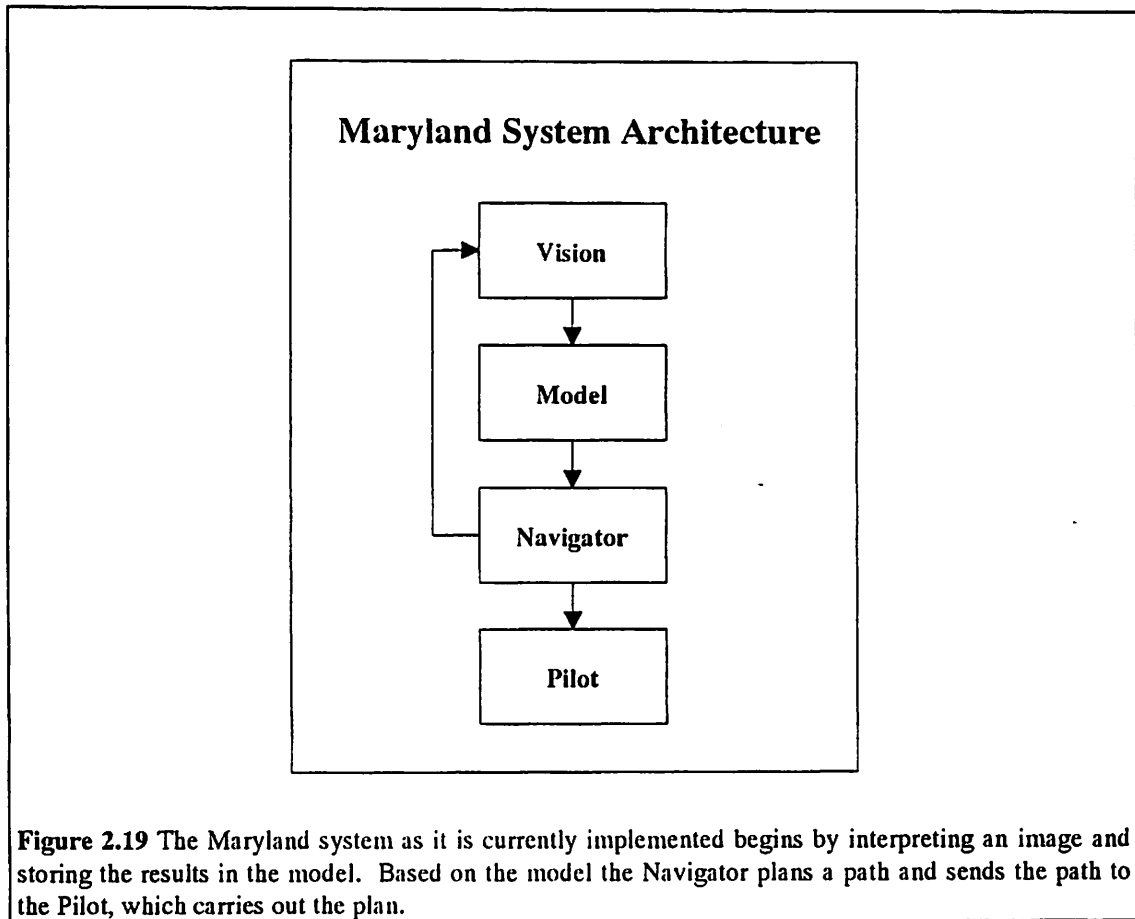
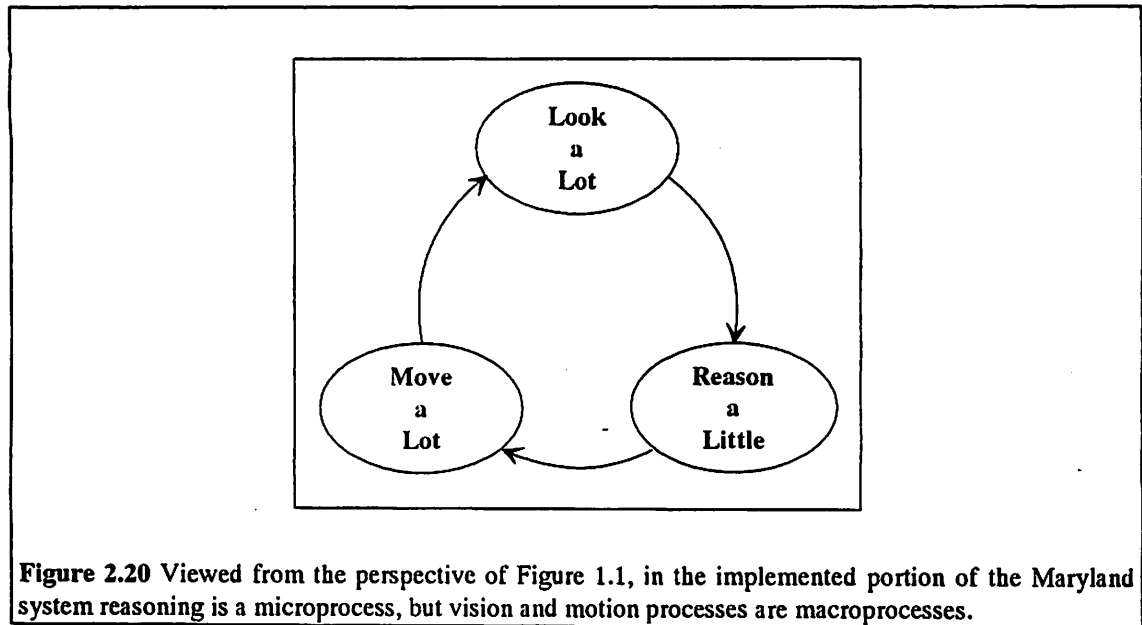


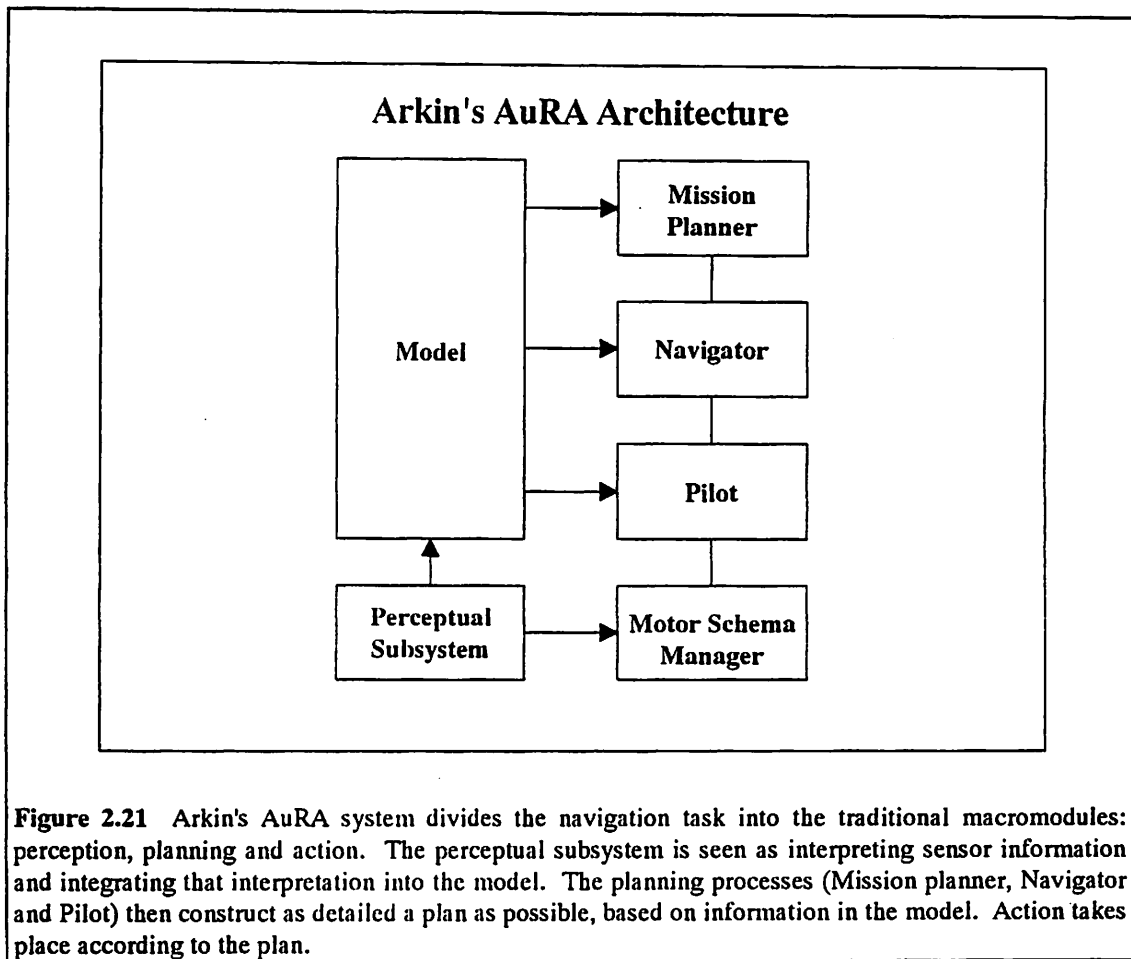
Figure 2.19 The Maryland system as it is currently implemented begins by interpreting an image and storing the results in the model. Based on the model the Navigator plans a path and sends the path to the Pilot, which carries out the plan.

The system, as it is currently implemented, is directed at following a road. The Planner module specifies a goal to reach a point (or distance down the road) and plays no further part in the system operation. This goal is passed to the navigator which invokes the vision system to find the road in the current field of view. The vision system (Figure 2.18) then performs a complete scene interpretation, looking only for road and non-road objects. Road objects are collected together to determine the road position and then the agent's location and heading are determined with respect to the road. The results of

this analysis is placed in the model. Using the information from the model, the navigator then plans a path down the road (only short range path planning is implemented), deriving a cubic arc which begins at the vehicle, slopes in the current direction of the vehicle heading and terminates 13 meters further down the road, intersecting and tangent to the road centerline. This path is then sent to the pilot, which converts this path into motion (Figure 2.19). This process continues, looping through the vision => model => navigation => pilot sequence until the goal has been reached. At present the system is limited to following an obstacle-free road.



The Maryland system breaks the intelligent process into the conventional modules of Figure 2.2 and, as Figure 2.19 indicates, it follows the information flow pattern shown in Figure 2.3. Vision, as indicated in Figure 2.18, is a macromodule in that it does a complete image analysis at each invocation. The final architecture for planning is not clear from the current design descriptions, but in the current implementation the Navigator plans paths which are designed only to keep the agent on the road and assumes that there are no obstacles. As implemented it qualifies as a micromodule, but it is not clear how it will develop. Motion takes place in 3 meter steps. Compared to the motion steps envisioned in the idea represented by Figure 1.1, this is a coarse-grain step. Figure 2.20 relates the Maryland system to the paradigm illustrated in Figure 1.1.



2.1.7 Ronald C. Arkin's AuRA architecture

Ronald Arkin's AuRA system [Arkin, 1987], a predecessor to the work described in this dissertation, was designed to perform intelligent navigation using "Harvey", the same mobile robot described in Chapter 1. The architecture for this system is illustrated in Figure 2.21. The role of perception in AuRA was seen as resulting in a "collection of plausible hypotheses and interpretations of sensor data with associated confidence levels that reflect their uncertainty."⁵ This interpretation was used to update AuRA's model of the environment.

The environmental model in this system is divided into three parts: one for vision, one for navigation planning and another for piloting. The vision model, used by the perceptual subsystem, is a schema based model, incorporating is-a and part-of hierarchies [Weymouth 1986]. The navigation planning

⁵ Arkin, page44.

model is a two dimensional model, called a "meadow map", of the surface to be traveled. This model represents space as a collection of convex polygons, the "meadows", each representing an area which is known to be free of obstacles. These meadows were computed from a polygonal representation, entered by hand, of the known obstacles in the environment. The pilot representation models the environment as a potential vector field over a two dimensional surface. Each point on the surface is assigned a vector which indicates the direction and speed the robot should have at that point.

AuRA used a hierarchical planning scheme which divided the planning effort into three levels: Mission planner, Navigator and Pilot (Figure 2.21). The mission planner would convert general problem statements into, among other things, navigational goals. This function is similar to what was referred to as "the problem solver" in Chapter 1. The navigator, or "global path planner", accepts a navigational goal and converts it into a complete piecewise linear path. This path was assumed to be developed to the point that it would avoid all known (modeled) obstacles. The pilot assumes that the plan generated by the navigator is complete and correct. Its primary objective is avoidance of unmodeled obstacles.

Perception in the AuRA system, like the systems described above, was envisioned as a subsystem that models the environment. The long term strategy for this work was to use the University of Massachusetts VISIONS system [Draper, Collins, Brolio, Hanson and Riseman, 1989] to provide a complete interpretation of the image, guided by the agent's goals. This interpretation was then to be used to update the navigator and pilot models. As implemented, sonar was used to detect obstacles and vision was used to recognize and locate pathways. Each detected obstacle was modeled as a potential field (a component of the pilot model) with vectors directed away from the detected surface. Pathways were modeled as fields of vectors pointing to the center of the path.

Path planning performed by the navigator was done using the A* algorithm on a search space generated from points on the polygonal boundaries of the meadows. Arkin experimented with two sets of points: one set used only the midpoints of line segments, the other set added points "near" the endpoints of these segments. The output of the A* algorithm on these point sets was often less than optimal, so paths were then subjected to a path improvement algorithm. The final plan was a piecewise linear path from the start location to the goal. No complexity analysis was done in this work, but it was noted in

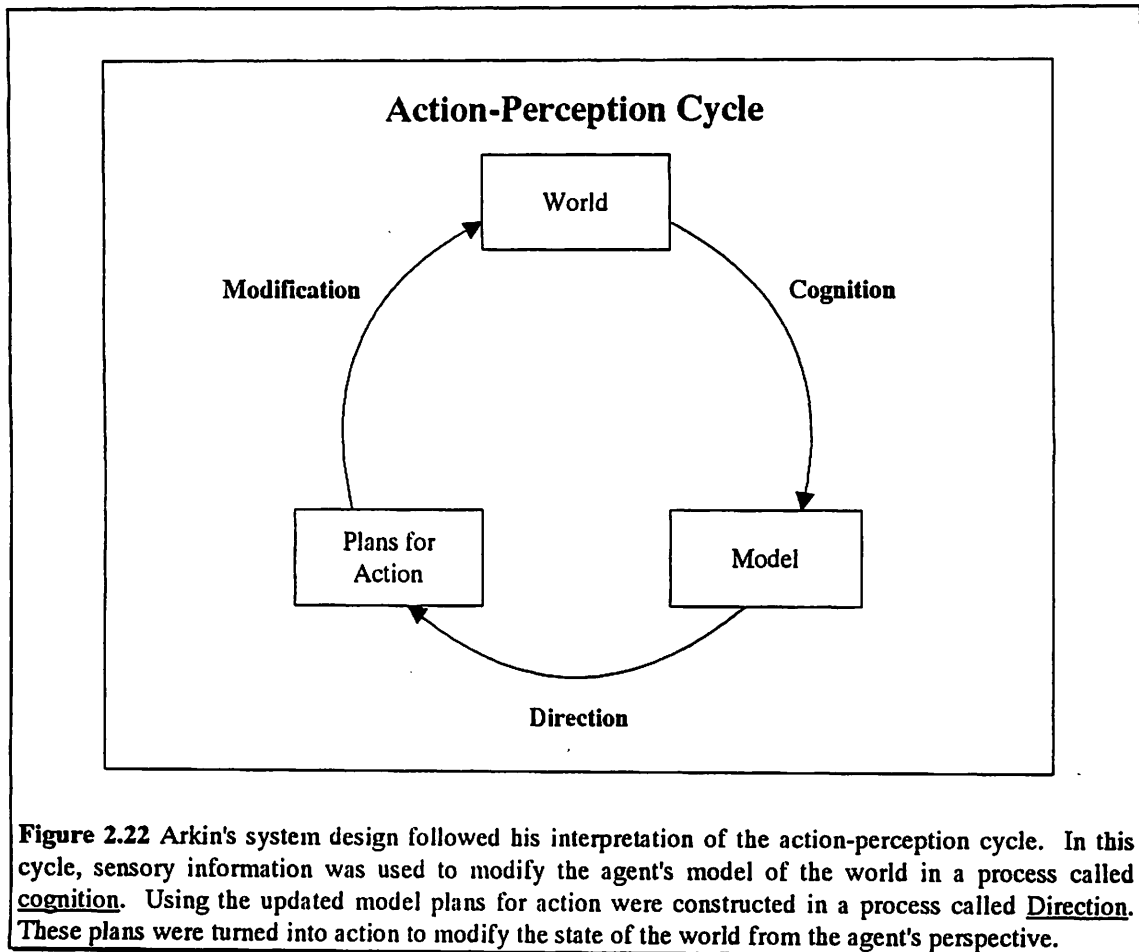
Appendix A [Arkin, 1987] that the time for the search using only the midpoints ranged from 0.46 to 1.38 seconds and that for the enhanced set ranged from 1.51 to 6.15. The ratio of the two longest times in these ranges is 4.5. The fact that this is greater than the factor of 3 in the number of points in the space is very likely due to the exponential growth of A* in the number of points in the model (see Chapter 5).

The pilot was invoked to carry out the plan generated by the navigator. For each linear segment of the plan the pilot generates a vector field which represents the goal of moving to the end of the segment. The goal position in this field has a "0" vector and vectors point toward that position. The vector fields generated by the perception module are added to this field and the pilot attempts to follow the "lines of force" in the resulting field. If the resulting trajectory deviates "too much" from the path segment specified by the navigator, the agent stops and the navigator constructs a new plan.

Arkin's work is another example of a macroprocess approach to intelligence. His system design followed what he called the "action-perception cycle" illustrated in Figure 2.22. In this view Cognition was a process of interpreting a sensory input; the VISIONS system was envisioned for this purpose. Direction was the process of creating an entire plan of action; A* was used for this purpose. Finally, the plans were transformed into actions. Errors or surprises discovered after an action would result in reinvoking A* to create a new plan.

Recognizing that the amount of computation in this loop was very large (NP complete?) Arkin's approach was to specialize perception for each activity, doing what he called "action-oriented perception". When traveling down a road, for example, the perceptual processing need not interpret everything in the image. It need only find obstacles and the road. This could be accomplished using parallel processes: an obstacle finder and a road finder. Each process was simpler and the processes could be performed concurrently. These processes were used as part of the pilot process. The results of these perceptions were used to create the vector fields used to control action. The resulting system still has a macroprocess architecture, however. Limiting sensory interpretation to finding a road or an obstacle is simpler but is still a coarse grained task. Like the Carnegie Mellon approach, the strategy is to divide the sensory interpretation task into pieces which can be executed simultaneously to reduce

execution time, but the pieces are still macroprocesses. Arkin's work showed an ingenious method for integrating the results of these perceptions.



2.1.8 Other Systems

Sections 2.1.1-2.1.7 describe projects which have implemented or designed complete autonomous robot systems. There are other projects which have done important and influential work on portions of the navigation problem. A few illustrative examples include the work done by the research teams associated with Martin Marietta, Dickmans and Graefe, and Faugeraus.

The Martin Marietta project focused its efforts on the problem of enabling a mobile robot to steer its way along a road [Turk, Morgenthaler, Gremban, and Marra, 1988]. The mobile robot, "Alvin", was a 16,000 pound vehicle equipped with a color video camera, a laser range scanner, a Land Navigation System (LNS), a VICOM image processor, and several other computers. In 1986 Martin Marietta

demonstrated Alvin's ability to successfully traverse a 2.6 mile test track at speeds up to 6.25 miles per hour.

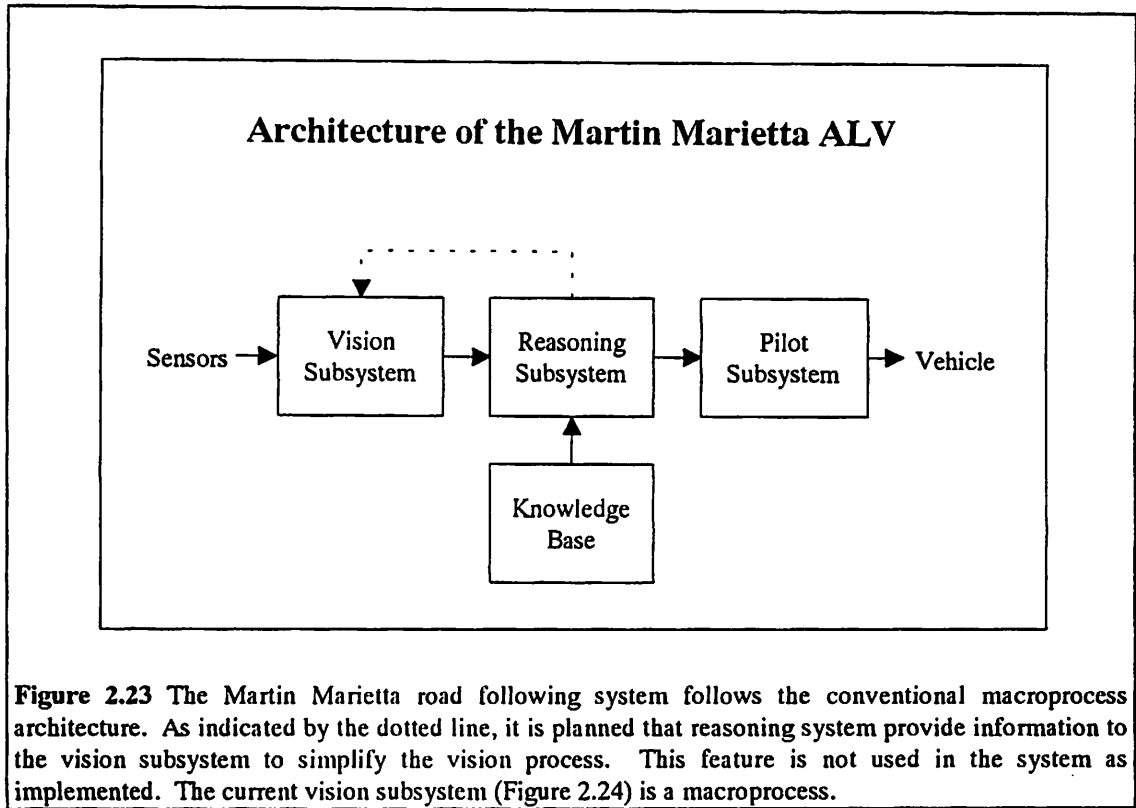


Figure 2.23 The Martin Marietta road following system follows the conventional macroprocess architecture. As indicated by the dotted line, it is planned that reasoning system provide information to the vision subsystem to simplify the vision process. This feature is not used in the system as implemented. The current vision subsystem (Figure 2.24) is a macroprocess.

The architecture of the Martin Marietta ALV road following system consisted of a knowledge base and three subsystems: a Vision system, a reasoning system and a pilot subsystem (Figure 2.23). The vision subsystem (Figure 2. 24) built a 3D model of the road in front of the vehicle. Beginning with a digitized image, the first step in this process was segment the image into regions by classifying each pixel as road or non-road. The next step was to extract the boundary of the road regions and select points which represent the left and right hand edges of the road. These 2D road edge points were then transformed into 3D points using camera parameters and some assumptions about the geometry of the traveling surface. These 3D points represent a model of the road.

The reasoning subsystem uses the 3D model information from the vision module to create a trajectory for the agent to follow. Even with the VICOM processor, the vision module was unable to create 3D models quickly enough to be used for steering the vehicle directly, so the reasoning module

pieced together several 3D models to develop a smooth trajectory for the agent to follow. The pilot controlled the motion of the vehicle by comparing the position determined by the LNS with the trajectory developed by the reasoning module.

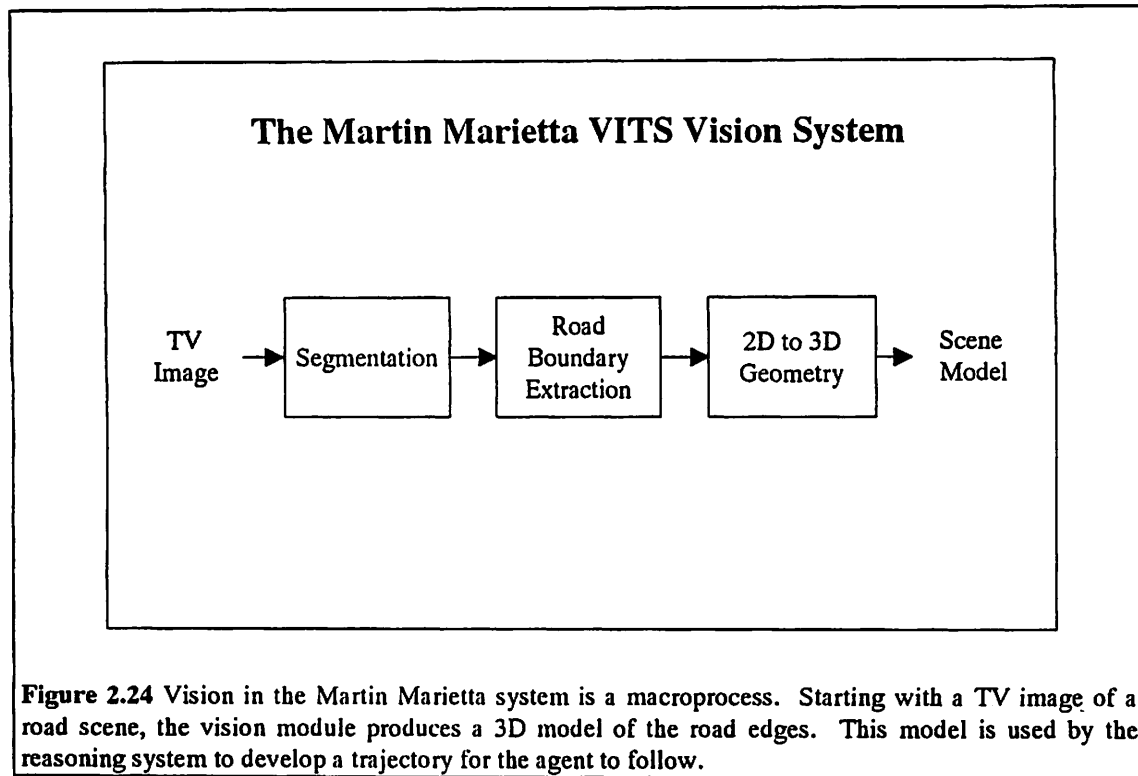


Figure 2.24 Vision in the Martin Marietta system is a macroprocess. Starting with a TV image of a road scene, the vision module produces a 3D model of the road edges. This model is used by the reasoning system to develop a trajectory for the agent to follow.

A research group at the Universität der Bundeswehr München has constructed several vehicle control systems based on computer vision [Dickmans and Graefe 1988a, 1988b] and [Dickmans, 1990]. The most impressive of these systems is able to drive a vehicle along a (controlled) section of the Autobahn at speeds of over 60 miles per hour (100 km/hr). This work is based on a system design which uses image sequences to determine the motion of the vehicle. As the vehicle moves, features of known objects are located in each image and their location is compared to what is calculated based on a set of differential equations which approximate the vehicle's motion. Differences between the calculated and observed feature positions are used to modify the equations. After a number of images have been processed the motion approximation stabilizes and a *measure* of the vehicle's motion is obtained. Servo controls are actuated by the system to modify the *actual* motion of the vehicle until the measured motion parameters satisfy the conditions of the problem. For road following the condition to be satisfied is that the vehicle

stay in the center of a lane. This aspect of their work may serve as an alternative way to accomplish action-level perceptual servoing described in Chapter 6. This work has not as yet developed to the point of specifying a completely autonomous system. Their technique does not address planning or how plans and actions would be reconciled.

The research team at INRIA has used sophisticated vision algorithms involving stereo to build a model of an office environment [Toscani and Faugeras 1987] and [Zhang and Faugeras, 1990]. The modelling process builds a 3D line model obtained from a sequence of stereo pairs. As each stereo pair is analyzed, the resulting model is used in a Kalman filtering process to refine and the robot is moved to a new position. This work has not yet developed to a full navigational system, but in [Zhang and Faugeras, 1990] their view of an intelligent system consists of three modules called: perception, Intelligent Decision and Control, and action. This is the standard macromodule model of Figure 2.2. The work described to date complements the accomplishments described in this dissertation. It addresses the model building and updating capabilities identified in Figure 1.2.

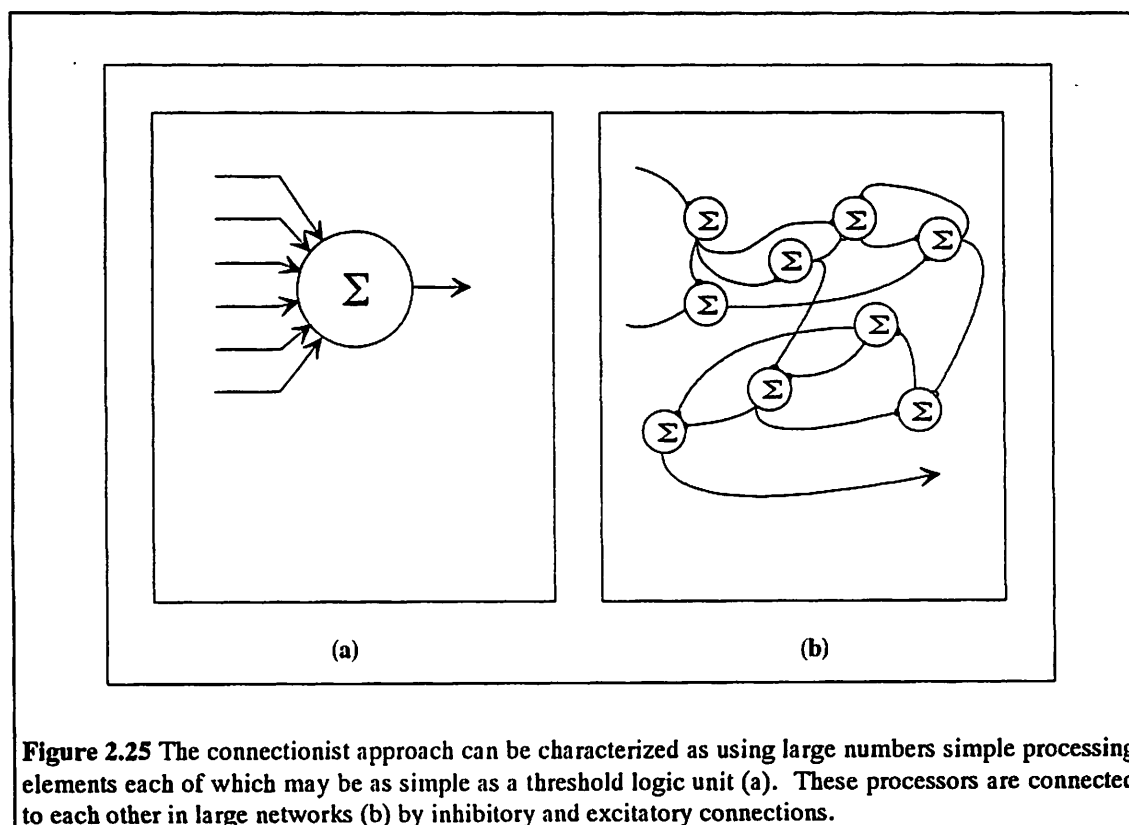


Figure 2.25 The connectionist approach can be characterized as using large numbers simple processing elements each of which may be as simple as a threshold logic unit (a). These processors are connected to each other in large networks (b) by inhibitory and excitatory connections.

2.2 The Connectionist Approach to Artificial Intelligence

Unlike the symbolic AI approach, which views intelligence as a process operating on a set of symbol structures, connectionism views intelligence as a phenomenon which emerges from the interaction of large numbers of very simple processing units. This concept is perhaps older than the "symbolic approach". Some of the earliest theories of artificial intelligence fall into this category [McCulloch and Pitts, 1943], [Selfridge, 1955] and [Rosenblatt, 1962]. Processors in these system usually take many inputs and produce a single output (Figure 2.25 (a)) and are connected to each other via inhibitory and excitatory connections. (Figure 2.25 (b)). Memory, rather than being represented in a centralized data structure, is distributed. It is represented by the configuration of inhibitory and exhibitory connections. Many of the well-known projects in this area are described in [Rummelhart and McClelland, 1988]. The next few subsections describes connectionist projects which relate to the issues explored in this dissertation.

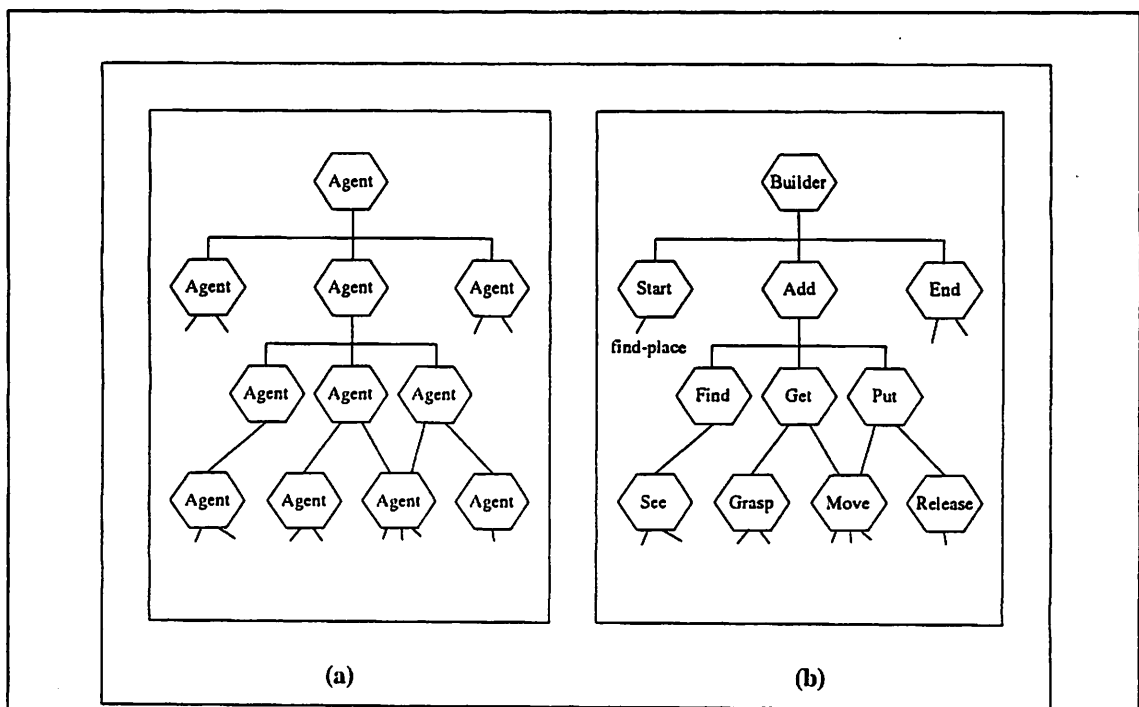


Figure 2.26 Minsky proposes that intelligence is accomplished by a large number of very unintelligent modules, called agents, organized into agencies which perform specific tasks (a). A tower builder, for example, would be an agency consisting of the large number of agents, some of which are exemplified in (b). None of these agents knows or does very much. The Builder agent merely sequences the Start, Add and End agents. The Add agent would in turn sequence the Find, Get and Put agents and so on. It is the agency which gets the task done.

2.2.1 The Society of Mind

About 17 years after writing a book about Perceptrons [Minsky and Pappert, 1969] which, according to Seymour Pappert [Pappert, 1988], was written with some hostility towards the connectionist community of the 1960's, Marvin Minsky described an interesting theory of intelligence with a strong connectionist flavor [Minsky, 1986]. This theory describes intelligence as arising from a very large number of modules, each doing a very specialized task. These modules, which he chooses to call agents, possess very little knowledge about anything and do very little by themselves; they contribute to knowledge and intelligent behavior via their connections with other agents (Figure 2.26 (a)). Each intelligent behavior is the result of the action of a collection of agents working together via these connections as per the example illustrated in Figure 2.26 (b). He gives these teams of agents the name agency and likens the organization to a society of agencies. Each agent can be a member of various agencies in the Society of Mind.

To illustrate the concept Minsky uses the example of a tower builder. The builder agency (Figure 2.26 (b)) has a builder agent which activates the start, add and end agents in sequence. The builder agent begins by activating the start agent. This agent is part of an agency which performs the task of finding a place to put the next block. When this task is done the builder agent activates the add agent. The add agent in turn activates the find agent, the get agent and the put agent. These agents accomplish their task by activating other agents (see, grasp, move and release) in their respective agencies to first find an appropriate block, then get it and finally put it in place.

The MAUV project, under a different interpretation than presented in section 2.1.4, can be viewed as an example of this idea. Each MAUV vehicle can be viewed as one of Minsky's agents and the collection of vehicles can be thought of as a Minsky society. The task of each vehicle is simplified in this society by its relationship to the other vehicles. These relationships distribute the effort, simplifying the task performed by each agent. This simplification makes it possible for each agent, and the society as a whole, to complete its task in real time.

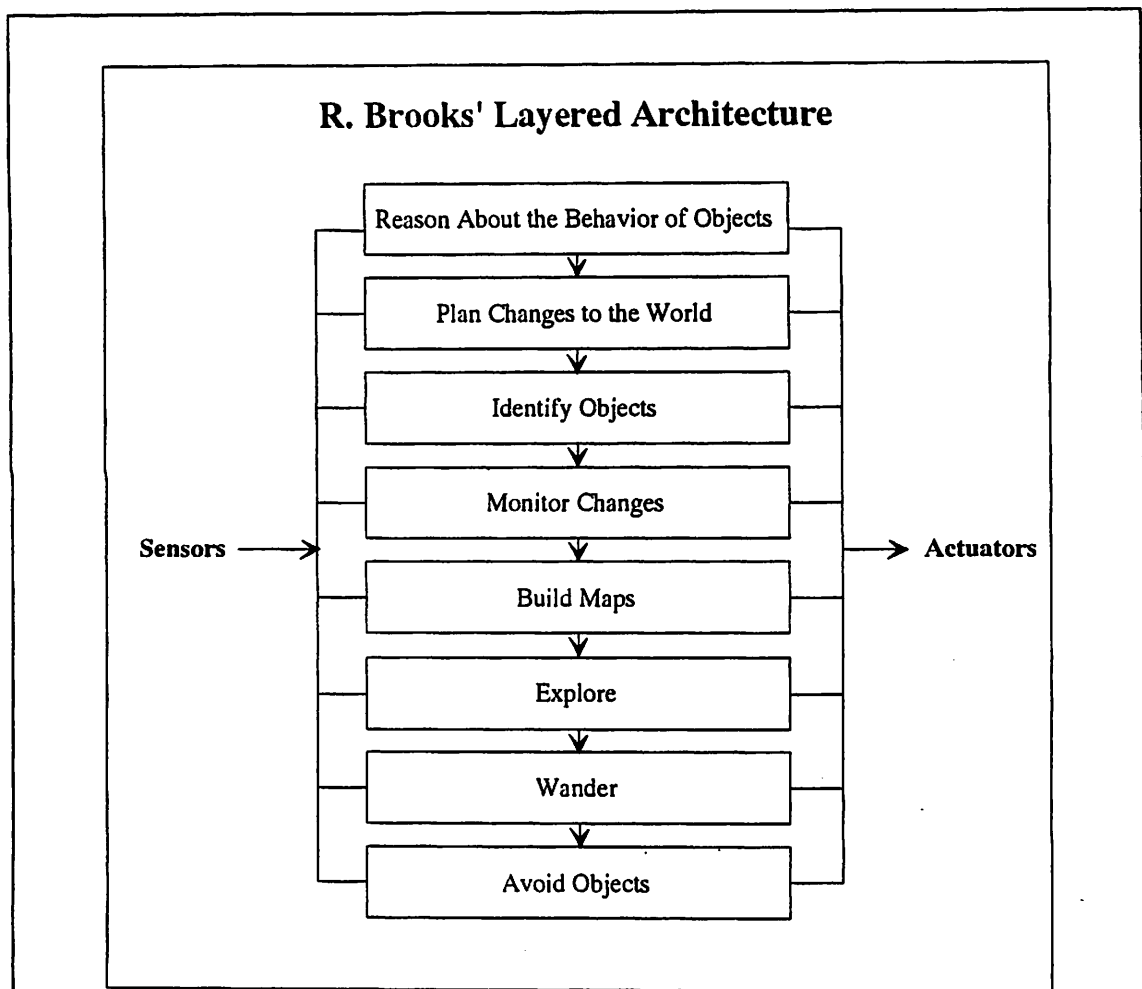


Figure 2.27 Brooks proposes a "layered" architecture. Each module in the above diagram is layered on top of the module below, using it to accomplish its task. Each module or "capability", as it is called, uses sensory information directly in ways which are specialized to the task that module implements. Each capability uses the modules below it by modifying the lower module's behavior. This modification is accomplished by overriding (or subsuming) signals in the datapaths of that module.

2.2.2 The Robots of R.A. Brooks

Brooks [Brooks, 1986] has taken issue with the architecture of Figure 2.2 and the information flow of Figure 2.3. The principal arguments against this arrangement relate to goal contention, multiple sensors and robustness. An agent will often have a multiplicity of goals it is trying to achieve and these goals can conflict. An agent must be able to suppress the less important of two conflicting goals and must be able to carry out nonconflicting goals in parallel.

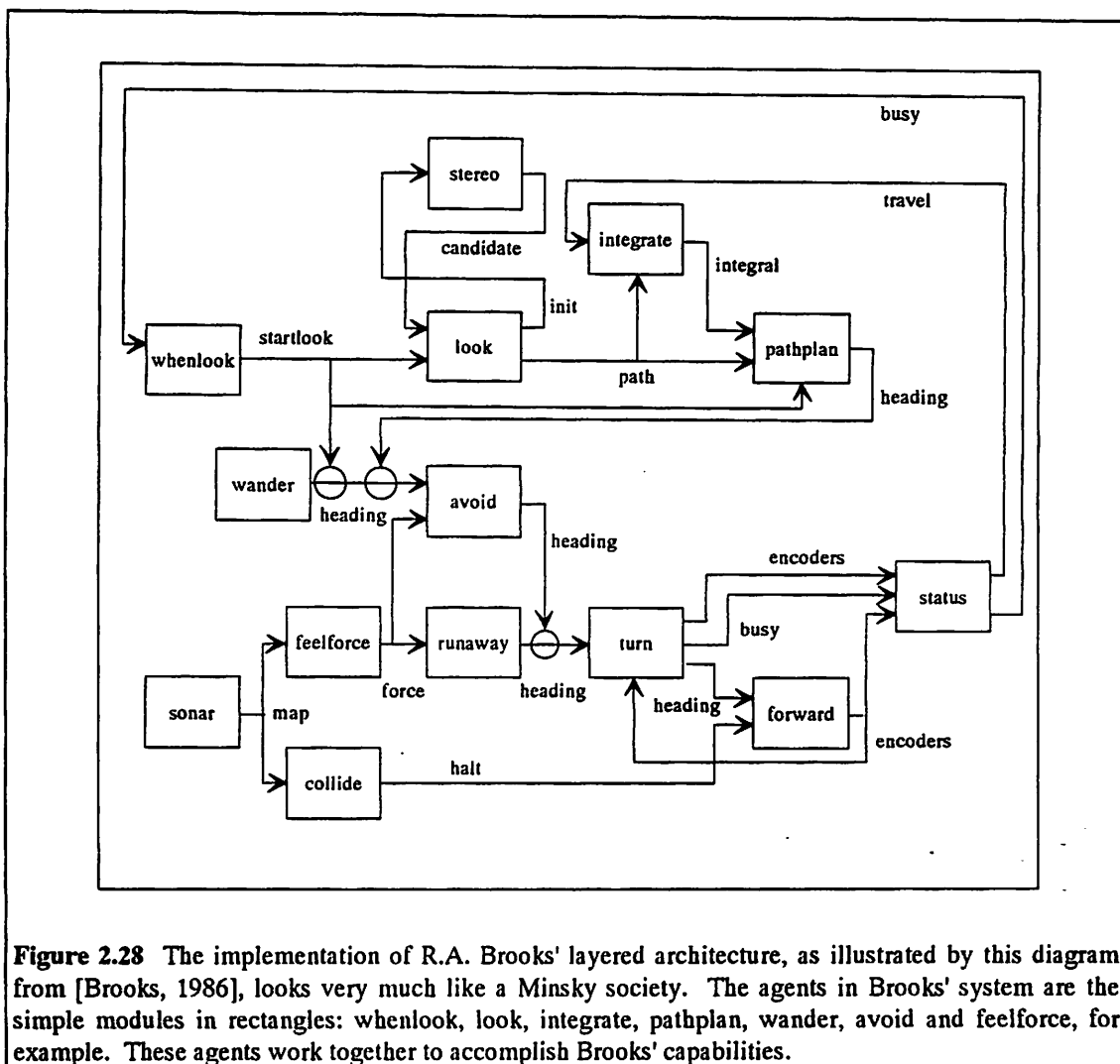


Figure 2.28 The implementation of R.A. Brooks' layered architecture, as illustrated by this diagram from [Brooks, 1986], looks very much like a Minsky society. The agents in Brooks' system are the simple modules in rectangles: whenlook, look, integrate, pathplan, wander, avoid and feelforce, for example. These agents work together to accomplish Brooks' capabilities.

Brooks proposes an architecture which decomposes control into behavior oriented tasks (Figure 2.27), rather than into the traditional mechanism oriented modules of Figure 2.2. The key concept is that each of the modules, or layers as they are better called, implements a level of behavior-related competence. Higher layers build on lower ones by examining data from and by injecting control into various places in the lower level layers. As explained, the idea is that a low level layer may implement an obstacle avoidance behavior. This behavior may use vision, sonar and touch sensors to accomplish its task. The point is that this layer functions at all times, striving to accomplish its task. This layer, in the absence of additional layer control, would cause the agent to begin moving away from an object which approached it and would cause the agent to stop if there were some object in its path. A second layer,

"on top" of this layer, could cause a behavior which could be described as wandering about without hitting any obstacles. The second layer would modify the behavior of the first by randomly introducing a new heading control signal at times when it senses that the lower level layer is not responding to an imminent collision. Additional layers would modify behavior further to explore, build maps, monitor changes, identify objects plan changes to the world and to reason about the behavior of objects.

The implementation of this idea, however, can be viewed as an ingenious interpretation of Minsky's society of mind. The layers of competence are in fact more like project milestones, reflecting the progress of Brooks work. At any one time the boundary between the layers are blurred to the point of extinction by the "interlayer connections" so that the system appears more as a collection of micromodules entitled: feelforce, collide, turn , runaway, avoid, forward and pathplan (Figure 2.28). This project has rather successfully implemented these Minsky-like ideas with a Brooks-like control structure to the point of constructing robots which wander about the halls. It is not yet clear, however, how this approach will deal with the higher level task layers such as planning and object recognition.

2.2.3 ALVINN, The Carnegie Mellon Steering Network

ALVINN [Pomerleau, 1990] is designed to be part of a steering subsystem for the NAVLAB mobile vehicle (described in Section 2.1.5). The ALVINN portion of this subsystem is a three layer neural network which takes visual input and outputs steering commands (Figure 2.29). One layer, the input layer, is a rectangular array of unit which form the "retina" of the system. These units activate according to how much red (r), green (g) and blue (b) light is incident on them. They activate according to the formula:

$$\text{Activation} = \frac{b}{255} + \frac{b}{r+g+b}$$

The second layer, the "hidden layer" consists of nine units. Each of these hidden units is connected to all retinal units. The third layer, the output layer, has 45 units each connected to every unit in the hidden layer. Each unit in this layer corresponds to a one of 45 steering commands from a spectrum which includes "steer hard left" at one extreme, "steer hard right" at the other extreme and "steer straight ahead" in the center.

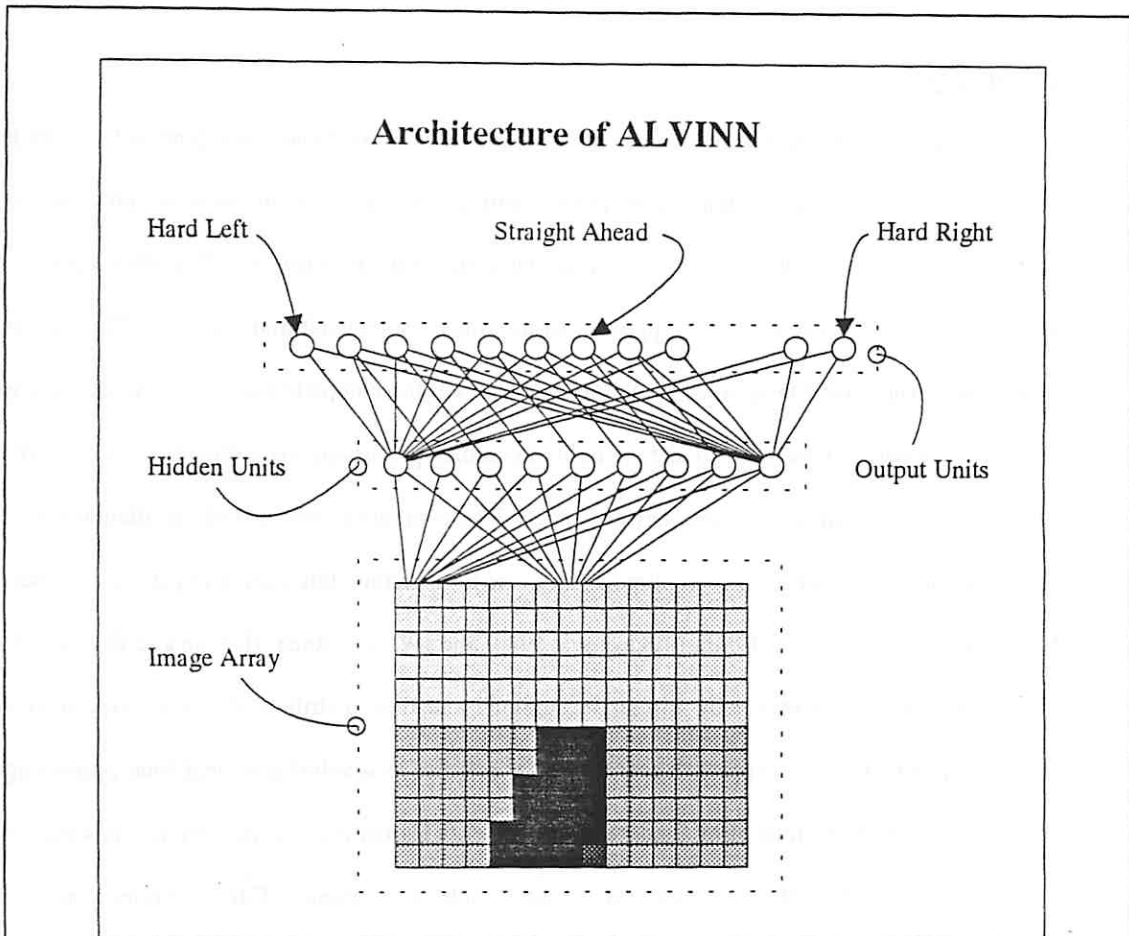


Figure 2.29 ALVINN is neural network consisting of three layers: a "retinal" layer, a hidden layer and an output layer. The retinal, or input layer, is a rectangular array of units, each acting as a pixel of the input image. The hidden units make the association between these retinal units and the units in the "output layer". Each unit in the output layer corresponds to a steering command such as "turn hard left".

Once trained, an input image of a road scene produces a pattern of activation on the output layer. The output unit with the largest activation is taken to represent the appropriate steering command. This network was shown to be able to process about 2 images per second, corresponding to a speed of 8 miles per hour. It was capable of imitating human driving over the section of road it was trained on and was able to generalize to some other parts of the same path. The experiments in [Pomerleau, 1990] were performed on a one lane test road under a variety of weather conditions. Performance on two lane roads or roads made of other surface types require retraining. Different road types result in very different representations. The challenge facing this program is to determine how to select the appropriate representation.

2.3 Summary

In reviewing the Symbolic and Connectionist approaches to Artificial intelligence it has been pointed out that symbolic approaches tend to be coarse grained systems built out of macroprocesses and the connectionist approaches tend to be fine grained built out of microprocesses. This distinction is one of practice, not theory. The symbolic approach as described by [Newell and Simon, 1972] can easily be interpreted to allow for fine grained symbolic systems. All the complete symbolic systems built to date, however, have been coarse grained and are built of complex processing macromodules which have not, as yet, been able to achieve real time performance. This community has turned its attention to parallel processing and the design of special purpose hardware to overcome this difficulty [Dickmans and Graefe 1988a, 1988b], [Choudhary, 1990], [Fukushima, 1990] and [Weems, Rana, Hanson and Riseman, 1990].

Connectionist theories are by nature fine grained and they distribute the processing over a large number of processes. This approach seems to offer promise as a solution to real time processing. The robots of R. Brooks seem to offer support to this hope. This community has not yet, however, offered any proposals which relate to intentional activity, such as planning. Current projects address only reactive behavior.

The approach described in the following chapters differs from what has been done by either of these two schools. It was originally motivated by the symbolic approach: it is consistent with Newell and Simon's information processing theory. It differs, however, from symbolic AI *practice* in that it is not a coarse grained system constructed of macroprocesses. Instead, reason, action and perception are executed as microprocesses and their invocation is tightly interwoven. Each invocation of these microprocesses does only a little to affect the overall intellectual process. Intelligent activity takes place over many invocations. These microprocesses bear some similarity to the connectionist processing elements or Minsky's "agents". Their role in the system differs, however, in that they are executed one (or maybe a few) at a time and their invocation is directed by an overall goal. The system is goal directed, rather than "reactive" in the sense that connectionist systems are sometimes described. Intelligent activity in this system does not emerge. It develops because the system is attempting to attain

a goal. The tight interweaving of microprocesses, however, allows the system to react to its environment in service of that goal.

The next chapter, Chapter 3, describes the ideas of this dissertation in more detail, beginning with an abstract description and concluding with a system overview. The remaining chapters discuss implementation details and experimental results.

CHAPTER 3

THEORY AND THE EXPERIMENTAL IMPLEMENTATION

As we walked through the Berkeley campus, Kingsley, who had been blind from birth, tapped his cane here and there, checking his path. Then, suddenly, he stopped for a second, tapped his cane lightly against a nearby tree, turned his head and pointed towards the tower to our right and said: "The tall building over there is called Campanelle."

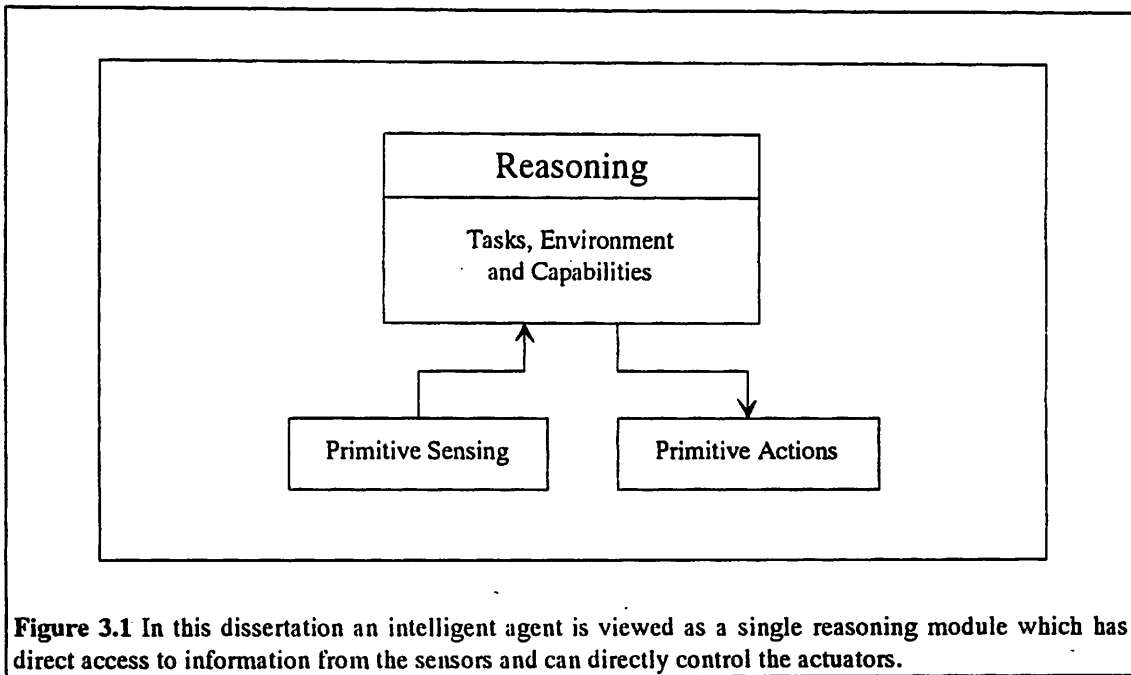


Figure 3.1 In this dissertation an intelligent agent is viewed as a single reasoning module which has direct access to information from the sensors and can directly control the actuators.

This story is one of the motivating incidents that led to the design of the system architecture described in this thesis. It is a story of an intelligent agent that reasons about what he is doing and uses his receptors (together with his cane) to check that his reasoning is consistent with reality. Kingsley can be viewed as a concrete example of the "reason-a-little, move-a-little, look-a-little" (RML) paradigm described in chapter 1. Unable to model every detail of his environment or to "see" what lies ahead, Kingsley would not construct plans in great detail before he began to walk. His plans would only be sketched in enough detail that he could decide which step to take next. After each incremental motion he would use his cane to verify either that the path was clear or that a landmark was where he expected it to be. Kingsley's perception of his environment would not develop as a result of a single sensory analysis by a macroprocess; it would be the result of many iterations of his reasoning, motion and perceptual microprocesses in an RML loop. The central claim of this dissertation is that intelligent behavior in

general can be demonstrated by a system designed to operate in this way and that this design is efficient. This chapter begins by describing this design concept in detail (section 3.1), using the terminology and background developed in chapter 2 and concludes by introducing the details of a specific system, the experimental system, which operates according to this principle (section 3.2).

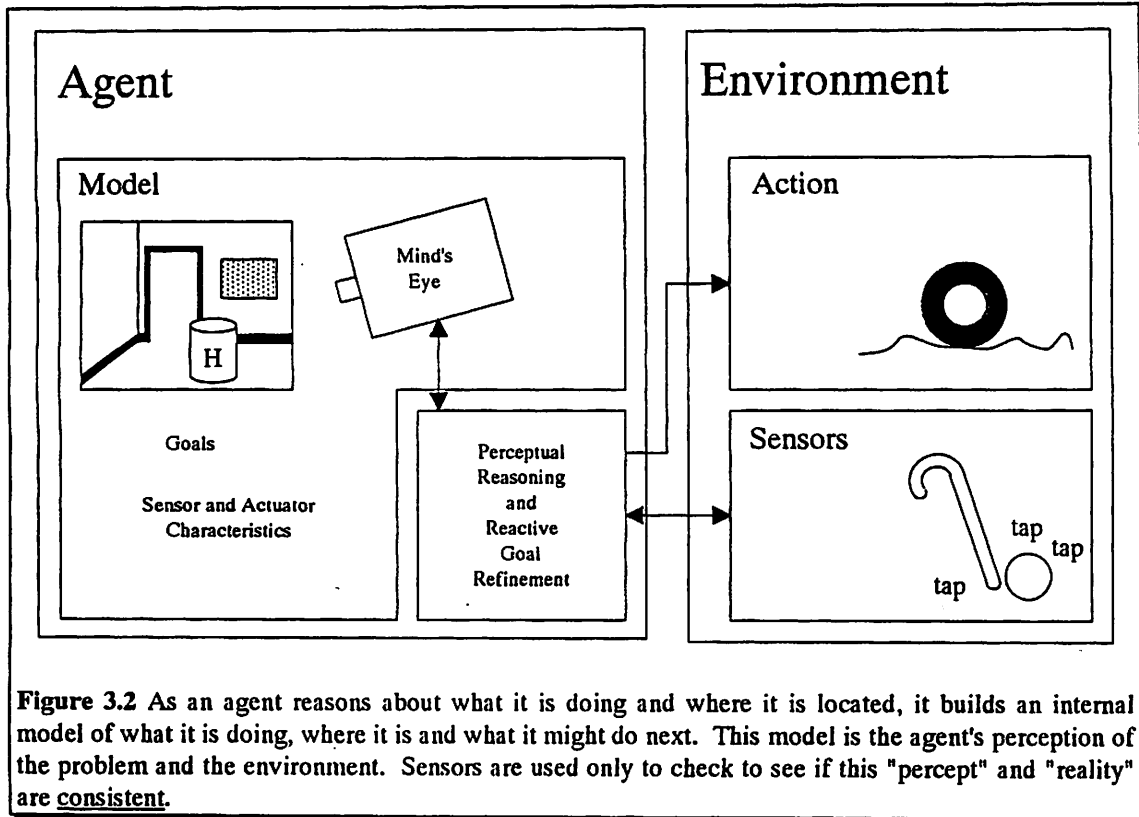


Figure 3.2 As an agent reasons about what it is doing and where it is located, it builds an internal model of what it is doing, where it is and what it might do next. This model is the agent's perception of the problem and the environment. Sensors are used only to check to see if this "percept" and "reality" are consistent.

3.1 The Theory

The story about the boatman (Chapter 1) and the story about Kingsley illustrate three important points about human behavior. The story of the boatman illustrates first that reason, perception and action effect each other during the course of carrying out a task and secondly that portions of the reasoning process are not developed until they are required. The second story illustrates the third point: that perception can be the result of very parsimonious, directed sensing (eg. the tap of a cane). These three points have been instrumental in the formation of the system philosophy developed in this dissertation. The RML system takes advantage of these behaviors. It is an unproven belief that their existence in humans is not accidental. The following chapters argue that these behaviors are efficient.

The terminology developed in chapter 2 makes it possible to describe the RML system philosophy more precisely:

1. Intelligent behavior can be demonstrated using symbolic AI techniques by exploiting an architecture which does not separate perception, action and reasoning into autonomous, intelligent, macromodules. The architecture of such a system would, rather, consist of a single reasoning module (Figure 3.1) which can access the sensors and the effectors directly. This single reasoning module reasons about the task, the environment and about the agent's capabilities.
2. One product of this reasoning process is an internal model of what the agent is doing, where it is and what it might do next. This model, which shall be called a "percept" in this dissertation, is the agent's current understanding of its environment, the task it is performing and where it is in the environment and in the execution of its task. Sensors are used to check to see if this "percept" and "reality" are consistent (Figure 3.2).
3. A second product of this reasoning process is the actions of the agent. The agent is successful when these actions have the net result of achieving the task.
4. The "percept" develops over time and is the result of a tight interweaving of processes traditionally identified as "reasoning", "perception" and "action". Using these traditional terms, "reasoning" is done just long enough to decide on a next "action"; the "action" is taken and then "perception" tests whether or not the result of the "action" is consistent with "reasoning" (Figure 3.3).

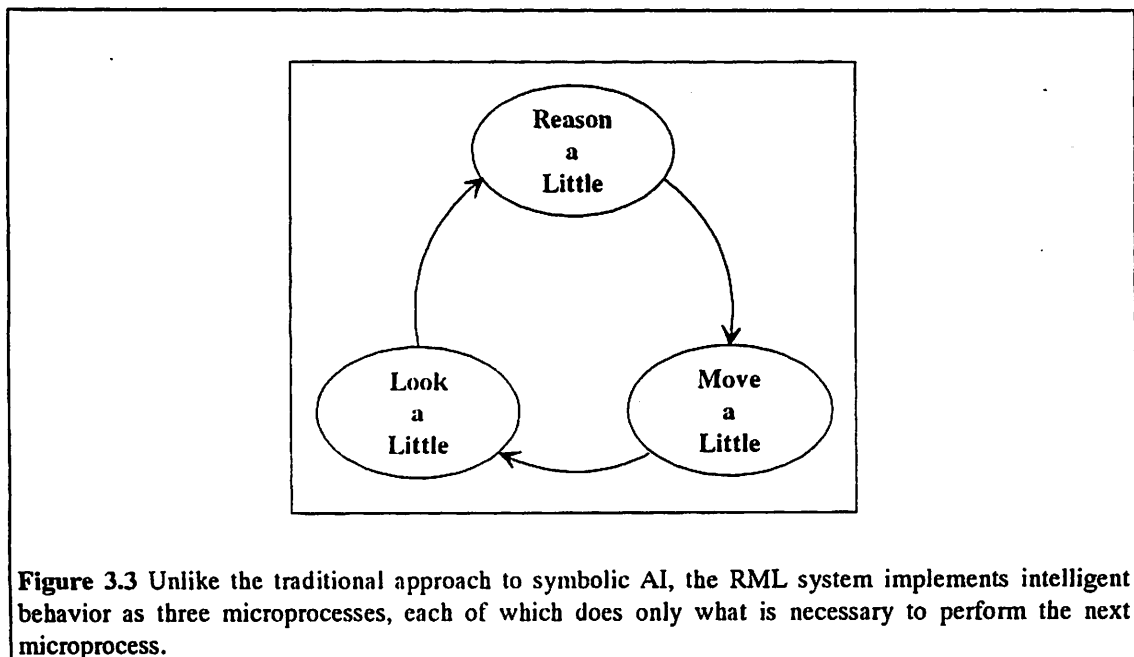


Figure 3.3 Unlike the traditional approach to symbolic AI, the RML system implements intelligent behavior as three microprocesses, each of which does only what is necessary to perform the next microprocess.

The information in a "percept" will depend on the current goals of the agent, the agent's sensors and on where the agent considers itself to be located in the environment. At the moment that the boatman and the savage "perceive" the oars on the beach the boatman's percept might include two long narrow shapes with handles, fittings for oar locks and relatively broad, flat ends (Figure 3.4). The percept of the

savage might include two long, gray, wood grained objects with less well defined shape. The percept of a jogger passing through may not include these objects at all. He may not "see" them.

One way the agent may verify the consistency of its percept with reality would be to project key portions of that percept onto a space which has properties similar to its arrangement of sensors, producing a "mental image", and then compare that image directly to sensor data. When using a visual sensor, for example, the percept might be projected onto a plane, forming something which has much in common with the mental images which have been the subject of intensive study by psychologists. These psychologists have built a case for their existence in humans [Kosslyn, 1980] and used them to explain the amount of time it takes subjects to recognize spatially rotated 3D objects [Shepard and Cooper 1982]. No claim is made in this thesis for the invention of the "mental imagery" concept, nor is there any claim that the use of it made here has any psychological justification. The phrase "mental image" does, however, provide a convenient analogy for communicating the ideas of this thesis. This concept emphasizes that the model used by the reasoning process includes sensor related information as well as abstract information. It is as if the agent creates a snapshot of the environment in its "mind". and checks this mental snapshot against its sensors. It is a summary of the collection of the facts collected from memory or indexed by the reasoning process to be used in solving the current portion of the problem.

There are advantages to this architecture. The interweaving tends to keep the agent's "percept" and "reality" in agreement, resulting in efficient reasoning and task execution. Each of the processes in the loop is made simpler by the others because they formulate specific, task related questions. The system as a whole tends to ask the right questions at the right time. The total process will tend to be more efficient than in the macromodule architecture for the following reasons:

1. Macroprocess architectures do detailed reasoning at each step. The "reasoning" macroprocess produces a detailed plan of action and the "perception" macroprocess does rather complete reasoning about the sensory array. This detailed reasoning is computationally expensive because it is unfocused and, moreover, it is likely to be wrong. The agent's model is likely to be incomplete or inaccurate; objects move; passages become blocked; and realized actions differ from the intended ones. Consequently, it is certain that this detailed reasoning will be modified, introducing additional computational expense. Reasoning can be simplified in the interwoven architecture because it can be done incrementally; as will be shown in the following chapters, it is possible to reason at a detailed level only as needed. Future details can be deferred so computation that would have been involved in developing them is not wasted.

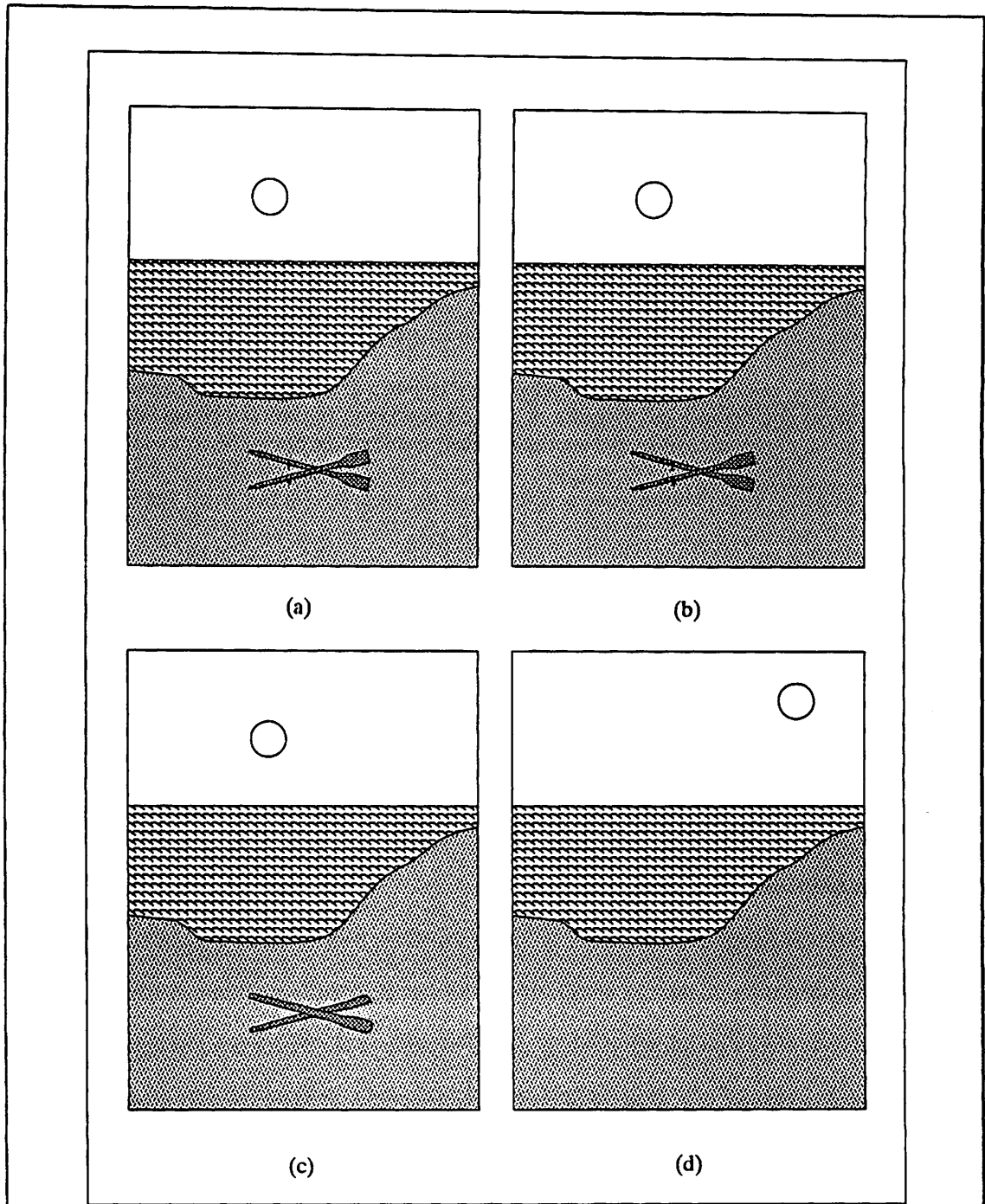


Figure 3.4 The environment of the story about the boatman and the savage (a) might be perceived differently by the boatman looking for his oars (b), the savage who is looking for firewood (c) and the jogger who had no interest in them (d). The boatman and the savage, who use the sun for navigation and telling time may perceive the position of the sun more accurately than the jogger who may only conclude that it is "a sunny day".

2. The tight interweaving of the microprocesses of the RML system makes it possible to check the consistency of reasoning and reality in a timely fashion. Differences can be detected early in the

intelligent processes so that actions, the "percept" and reasoning can be incrementally changed in such a way that modifications will tend to be minor.

3. In the macroprocess architecture, perception is the result of a separate reasoning system which is also expensive. The RML system architecture reduces duplication of computation by interweaving the reasoning activities in a way that makes it possible for each reasoning increment to pose more focused questions about the environment. This focus reduces computation.

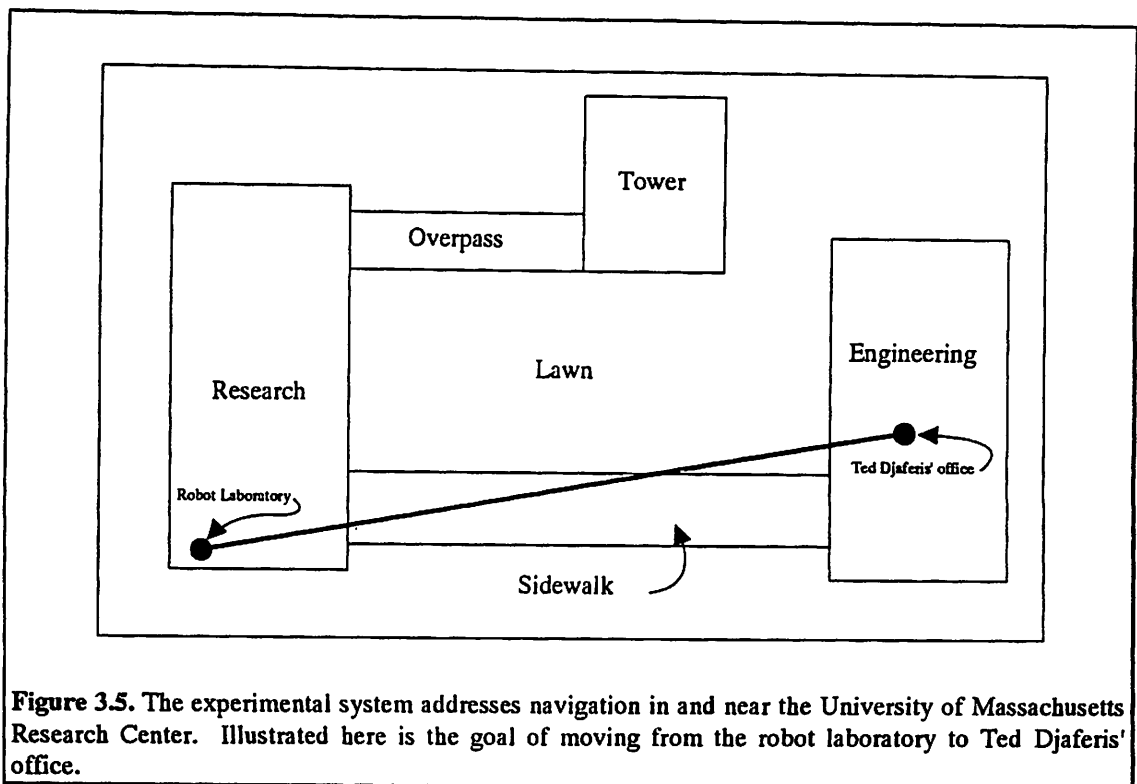
4. Execution of desired actions is inherently inaccurate and uncertain. Whenever an agent rolls along a surface, for example, the slipperiness or unevenness of the surface and the bulges in its tires may throw it off course. The result of each primitive action is rarely what its statement would predict. By interweaving, motor actions can be carried out more accurately; they can be continuously monitored and the results can be related to the plan.

5. Finally, the environment is not usually as remembered. Models are inaccurate. Not only does the environment change; but, because agents do not have an accurate global measurement of their location, the location of spatial entities and their relationships cannot be accurately remembered.

The RML system philosophy is consistent with the ideas present in Newell and Simon's definition of an information processor and with the ideas presented by the connectionists; but, as we saw in Chapter 2, symbolic AI research has generally followed the macroprocess approach and connectionist AI research has not yet implemented activities which might be called reasoning. The RML concept is different from current practice and promises considerable efficiencies. The remainder of this thesis describes one way to implement an RML system, in the context of the problem of autonomous navigation, and develops supporting arguments for the advantages to this approach.

3.2 The Experimental System

In order to investigate the RML concept an experimental system has been constructed. The design and implementation of this system illustrates the details of an example implementation of the RML concept and offers a platform for describing and exploring the concept in some detail. This system, as mentioned in chapter 1, is intended to provide the University of Massachusetts robot vehicle, "Harvey", with the ability to navigate about a portion of the campus: the Lederle Graduate Research Center and a portion of the outside environment (Figure 3.5). This problem addresses the application goal of this dissertation. Autonomous navigation is an important application problem and it has many important subproblems. This section offers an overview of the design. Before describing the experimental system it will be helpful to introduce some terminology.



3.2.1 Terminology

The navigation process, as discussed in chapter 1 (Figure 1.2), is initiated by the formation of a navigation goal. This is a statement of what navigational task is to be accomplished and is usually a goal to change location, as depicted in Figure 3.5 (other navigational goals will be described in Chapter 4). These goals are represented in conceptual dependency notation. The form:

(ptrans location-1 location-2),

for example, expresses the goal of moving from location-1 to location-2.

As the navigational system reasons about how to accomplish a goal it may decompose it into a number of subgoals. These subgoals are goals which, if all of them are achieved, will imply the achievement of the goal they decompose. Some subgoals need not be decomposed to be accomplished by the agent. These subgoals, called primitive subgoals or primitive goals, are goals for which the agent has a ready pattern of actions to solve. What is or is not a primitive subgoal depends on the capabilities of the agent. This concept can be made more precise after the introduction of the notion of a primitive action.

A primitive action is one of a finite number of basic actions that together can be used to describe every action the agent can take. A set of primitive actions can be thought of as a basis for its space of actions. The list of primitive actions for an agent will have at least one entry for each degree of freedom.

A human agent, for example, may have the following set of primitive actions:

- (agent-rotate-head degrees)
- (agent-bow-head degrees)
- (agent-tilt-head degrees)
- (agent-open-jaw degrees)
- ...

Harvey, the agent used in the implementation of the ideas of this thesis, has a vocabulary of only two primitive actions:

- (agent-move distance)
- (agent-turn angle).

Any action Harvey is capable of performing can be decomposed into some sequence of these two primitive actions.

The set of primitive actions associated with an agent may not be uniquely defined. There may be many such sets of primitive actions. Harvey, for example, can also be described by the following primitive action set:

- (agent-turn-positive)
- (agent-turn-negative)
- (agent-move-forward)
- (agent-move-reverse)
- (agent-stop-turn)
- (agent-stop-move)

This set offers easier description of "continuous" motions; but, due to the design of the hardware, Harvey's actuators operate in discrete steps, so the set of motions described by these two sets of primitive actions is the same. The first set was chosen here for its simplicity.

In terms of a set of primitive actions the notion of a primitive goal can be made more precise. Any goal which can be accomplished by a primitive action is a primitive goal. In addition, however, it may be that an agent has a script which describes how a set of primitive actions can be used to accomplish a particular goal. Goals of this type would also be primitive subgoals. Whenever the straight line path

between two locations A and B is unobstructed, for example, the goal (ptrans A B) would be a primitive subgoal for Harvey because the following script applies:

```
(script-move-unobstructed-straight-line-path (A B)
  (agent-turn (- (angle A B) current-heading)) ; turn towards B
  (agent-move (distance A B))) ;move the distance that separates A and B.
```

This script captures the idea that to traverse an unobstructed straight line path it is only necessary to turn toward the desired location and move along a straight line. In a sense a script of this type is a packaged plan for accomplishing a certain type of goal. The goal is primitive because the reasoning that went into creating the script need not be repeated.

As the agent moves through its environment it keeps track of where it is using landmarks. These landmarks are 3-dimensional entities or features which are associated with a particular location and which have characteristics which can be sensed by the agents receptors. An example landmark for Paris could be the Eiffel Tower. This landmark has world wide significance because there is only one on this planet. Other landmarks may only have local significance. The "door to the hallway" is significant in a particular room, but may be very ambiguous as a landmark when considered from a more global perspective.

When an agent decomposes a goal into subgoals it is said to be constructing a plan. In this dissertation the concept of a milestone is introduced as a means to measure plan progress. The term milestone, as used in project planning literature [Brooks, 1978], is used to describe a measurable event which can be used to determine that plan execution has reached a certain stage. In this work, the term milestone refers to a relationship that should exist between the agent and one or more landmarks after the accomplishment of a goal. For the goal

```
(ptans new-york paris)
```

one might associate the milestone

```
(in-front-of agent eiffel-tower 100 feet)
```

Milestones of this type can be verified by perceptual means.

As was pointed out in chapter 2, the practice has been that plans are developed in detail. In terms of the notion of a primitive subgoal this means that the term plan usually refers to a sequence of primitive

subgoals which solve the original goal. The RML concept does not develop plans in detail, so the notion of a plan sketch is introduced. A plan sketch is a sequence of subgoals, some of which may not be primitive, which solve the original goal. This concept will be discussed in more detail in the next section.

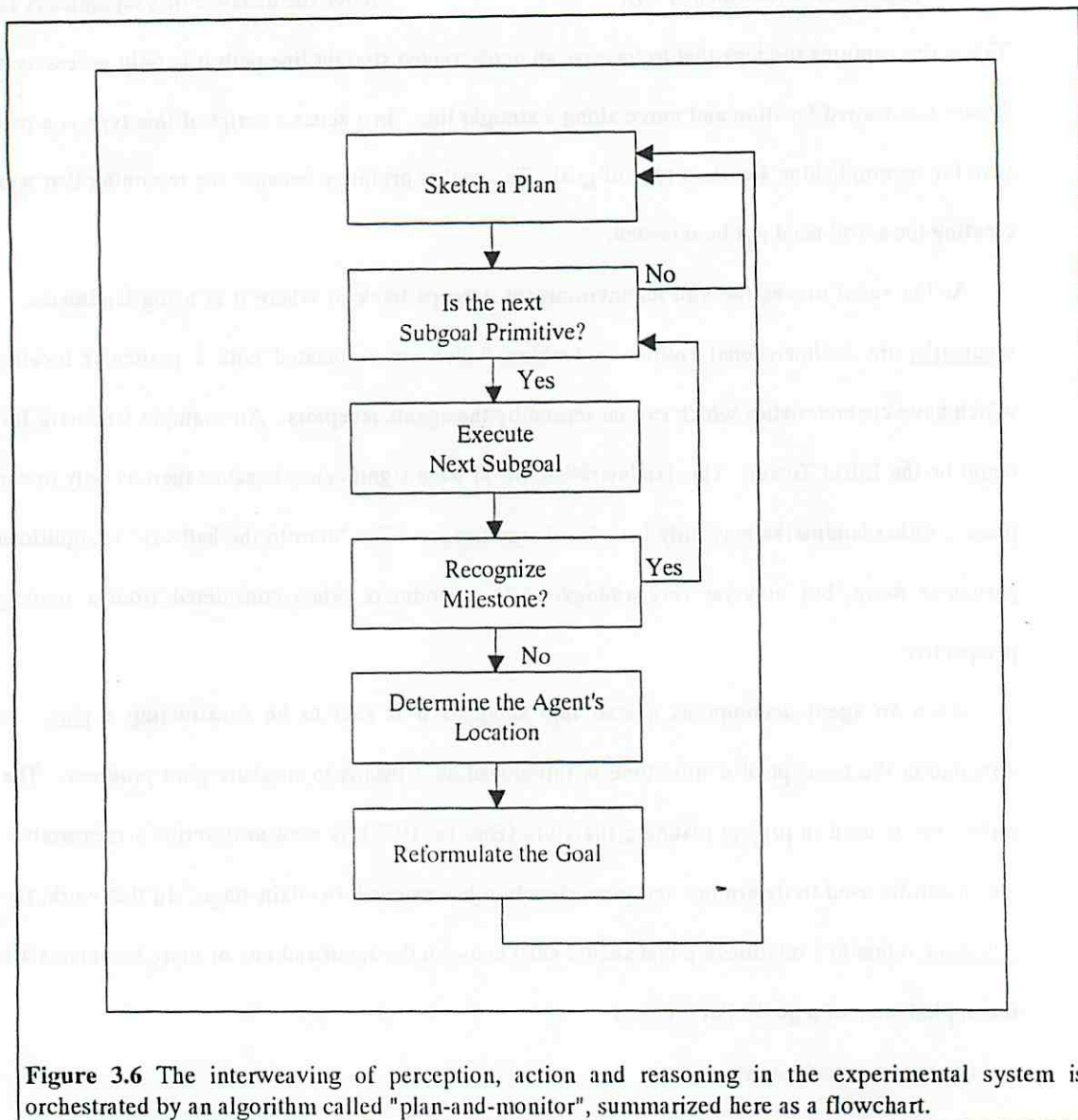


Figure 3.6 The interweaving of perception, action and reasoning in the experimental system is orchestrated by an algorithm called "plan-and-monitor", summarized here as a flowchart.

3.2.2 Plan-and-Monitor

In the experimental system the RML loop is implemented at three levels, two of which are part of the algorithm which provides the top level general control structure of the agent. This algorithm, called "plan-and-monitor", is defined in Figure 3.6; it begins with a goal

(ptrans location-1 location-2)

which it immediately converts into a plan sketch. The details of the plan sketching process will be described in Chapter 5, but Figure 3.7 (a) gives a pictorial example of a plan sketch for the goal shown in Figure 3.5. The goal

(ptrans robot-lab ted's-office)

has been decomposed into the plan sketch

(and

(ptrans robot-lab research-door-1)

(ptrans research-door-1 engineering-door-3)

(ptrans engineering-door-3 ted's-office))

This sketch and the associated goal decomposition tree are shown in Figure 3.8.

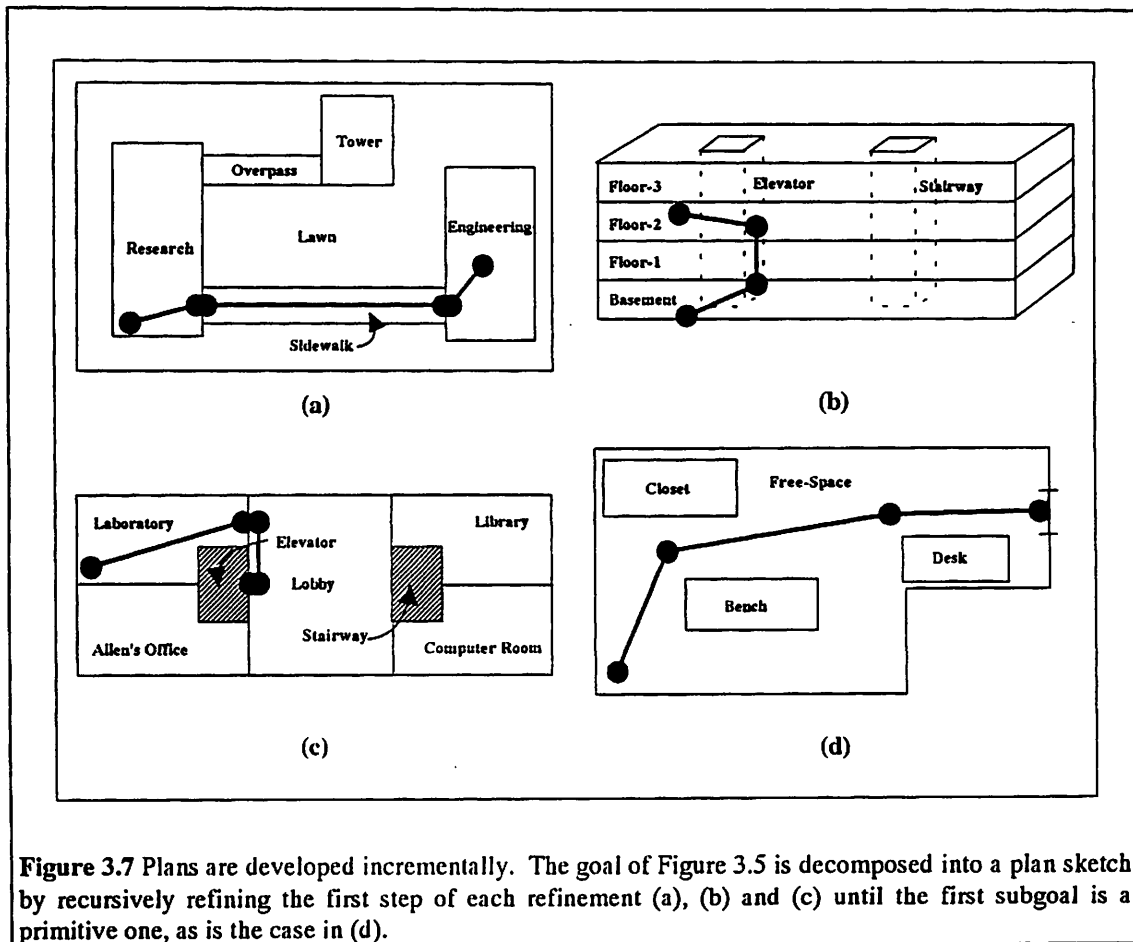


Figure 3.7 Plans are developed incrementally. The goal of Figure 3.5 is decomposed into a plan sketch by recursively refining the first step of each refinement (a), (b) and (c) until the first subgoal is a primitive one, as is the case in (d).

Having constructed this plan sketch, plan-and-monitor considers its first subgoal. If it is a primitive subgoal an attempt will be made to execute it. Primitive subgoals are accomplished using perceptual

feedback, in effect a simple version of the RML loop. Reasoning decides which primitive move to make (turn or move) and what perceptions should result from the action. The action is taken and a look determines whether or not the action had the desired effect. If not, a correctional action is taken. In this way the agent uses its sensors to stay on course. This process is described in detail in Chapter 6.

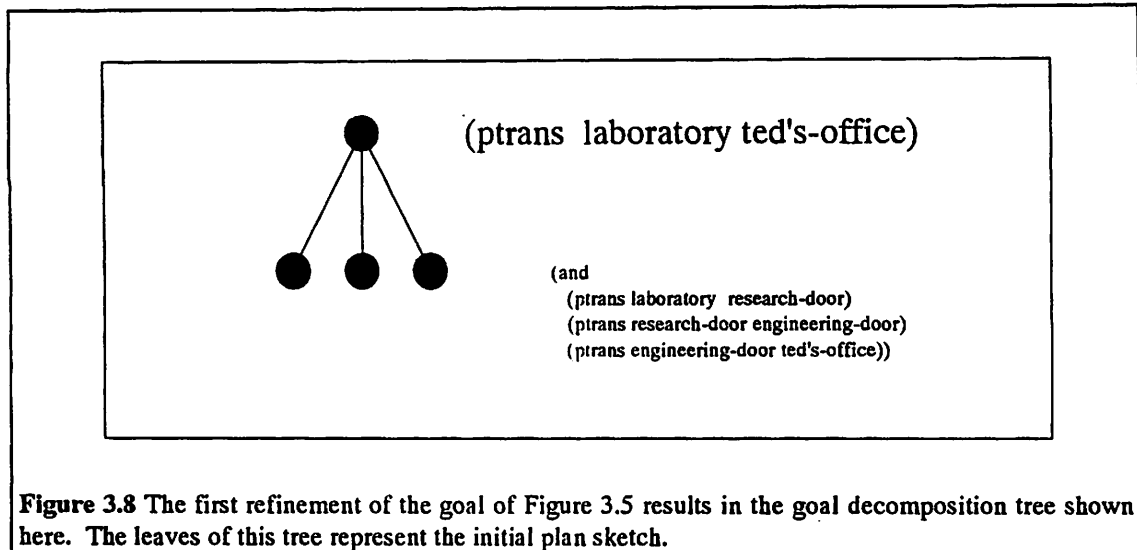


Figure 3.8 The first refinement of the goal of Figure 3.5 results in the goal decomposition tree shown here. The leaves of this tree represent the initial plan sketch.

Associated with each subgoal is a milestone. When the subgoal execution is complete, an attempt is made to recognize the milestone. In this process, described in chapter 6, the landmarks of the milestone are identified and their relationship to the agent are determined. If the relationship is within tolerances the milestone is said to be recognized. This means that reason (as reflected in the plan sketch) and reality are consistent so the next subgoal is considered.

When a milestone is not recognized the agent is lost. It must determine its location and construct a new plan sketch. The process of determining the agent's location, when it is lost, has not yet been implemented and is left as a matter for future research. Chapter 6 concludes with a discussion of this "where am I" process and how it might be implemented. The plan sketch reformulation would be similar to constructing the original plan sketch.

When the first subgoal in a plan sketch is not primitive, plan-and-monitor is invoked recursively to form a more detailed plan sketch for the first subgoal. In the example of Figure 3.7 (a) the first subgoal

(ptrans robot-lab research-door-1)

is not a primitive subgoal, so it is refined to the plan sketch:

```
(and
  (ptrans robot-lab research-center-elevator-second-floor)
  (ptrans research-center-elevator-second-floor research-center-elevator-first-floor)
  (ptrans research-center-elevator-first-floor research-door-1))
```

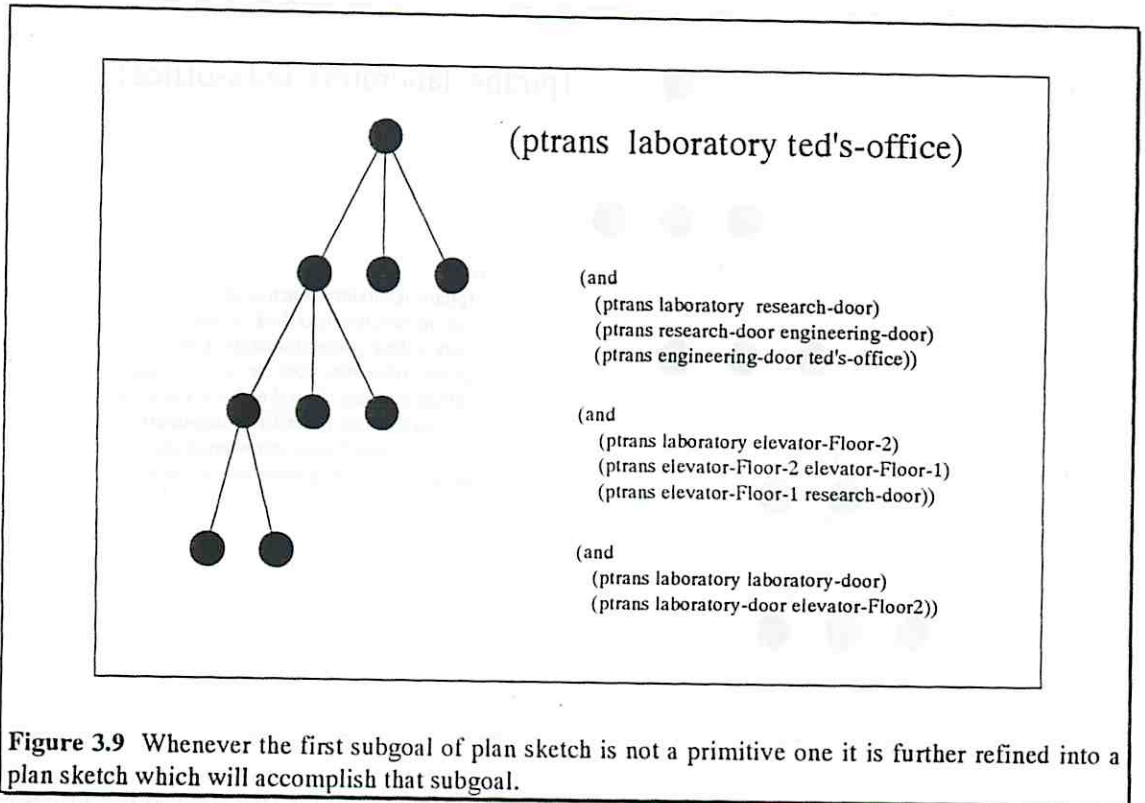


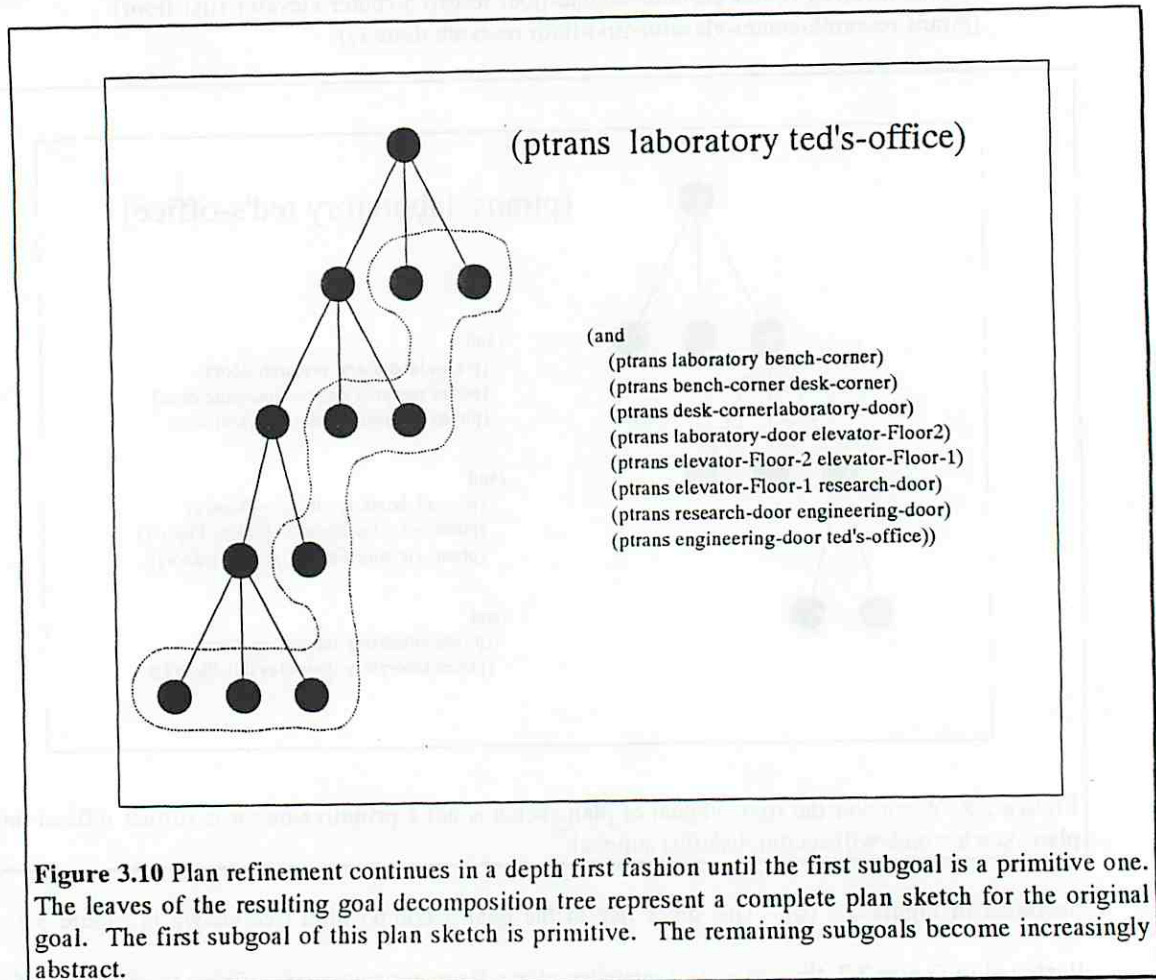
Figure 3.9 Whenever the first subgoal of plan sketch is not a primitive one it is further refined into a plan sketch which will accomplish that subgoal.

illustrated in Figure 3.7 (b). This gives rise to the goal decomposition tree shown in Figure 3.8. As illustrated in Figure 3.7, this process of recursive plan refinement continues, refining the first step of each plan sketch until the first subgoal is a primitive one. This is the situation shown in Figure 3.7(d).

The result of this process is a depth first refinement of the goal into the decomposition tree shown in Figure 3.9. The leaves of this tree form the current working plan sketch shown in figure 3.10. This plan sketch:

```
(and
  (ptrans laboratory bench-corner)
  (ptrans bench-corner desk-corner)
  (ptrans desk-corner laboratory-door)
  (ptrans laboratory-door elevator-Floor-2)
  (ptrans elevator-Floor2 elevator-Floor-1)
  (ptrans elevator-Floor-1 research-door)
  (ptrans research-door engineering-door)
  (ptrans engineering-door ted's-office))
```

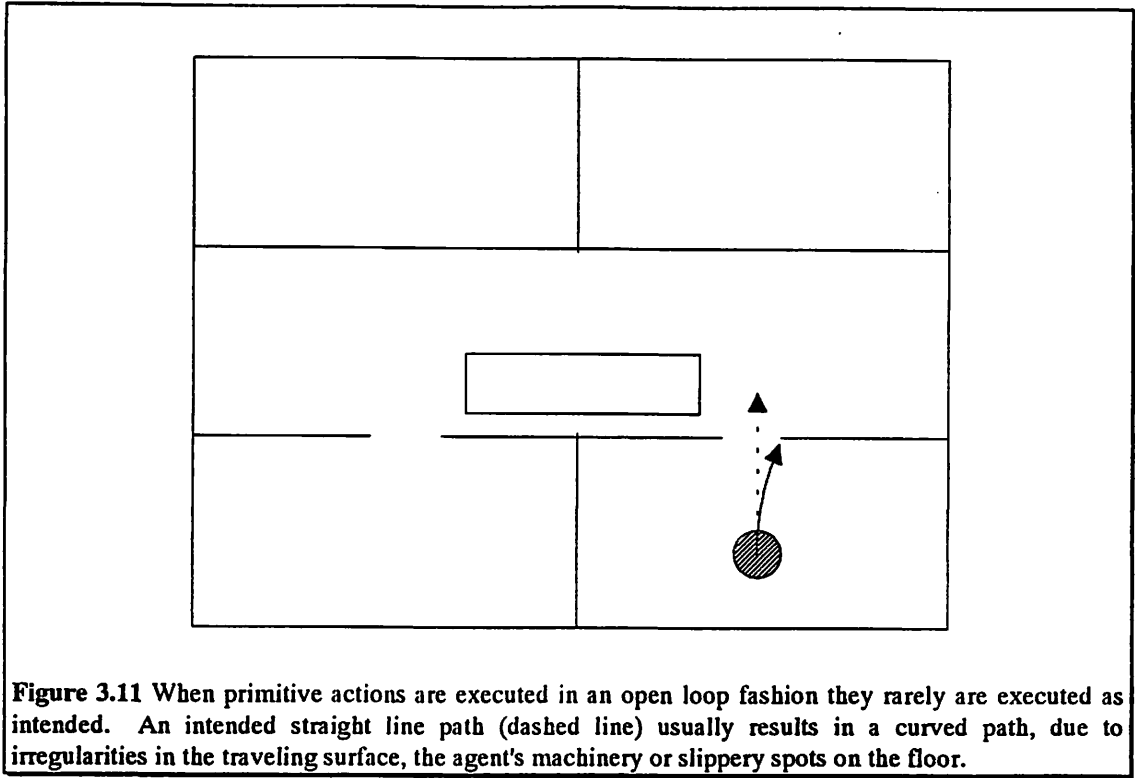
consists of only three primitive goals. The remaining goals describe goals which need to be decomposed before they can be acted upon.



As soon as the reasoning process has generated a plan sketch whose first subgoal is primitive action begins. Plan sketch detail is developed only as it is required. This keeps plan reformulation costs low, should they be required.

3.2.3 Perceptual Servoing: A different perspective on Plan-and-Monitor

The effect of plan and monitor is to implement three nested RML loops. These three loops use sensory information in different ways to keep the agent focused on attaining its goal, as viewed from different levels of abstraction. Because of the role perception takes in these loops they are called perceptual servoing loops. They are given the names: action-level perceptual servoing, plan-level perceptual servoing and goal-level perceptual servoing.



3.2.3.1 Action-Level Servoing

Action-level perceptual servoing serves to keep the agent "on track". The goal of this level of servoing is to maintain consistency between the agent's model of a primitive action and the realization of that action. When primitive actions are executed in an open loop manner they rarely result in the expected effect (Figure 3.11). Action-level servoing uses the perception of landmarks to detect differences between the intended motion and its realization and uses these differences to determine what correctional actions to make.

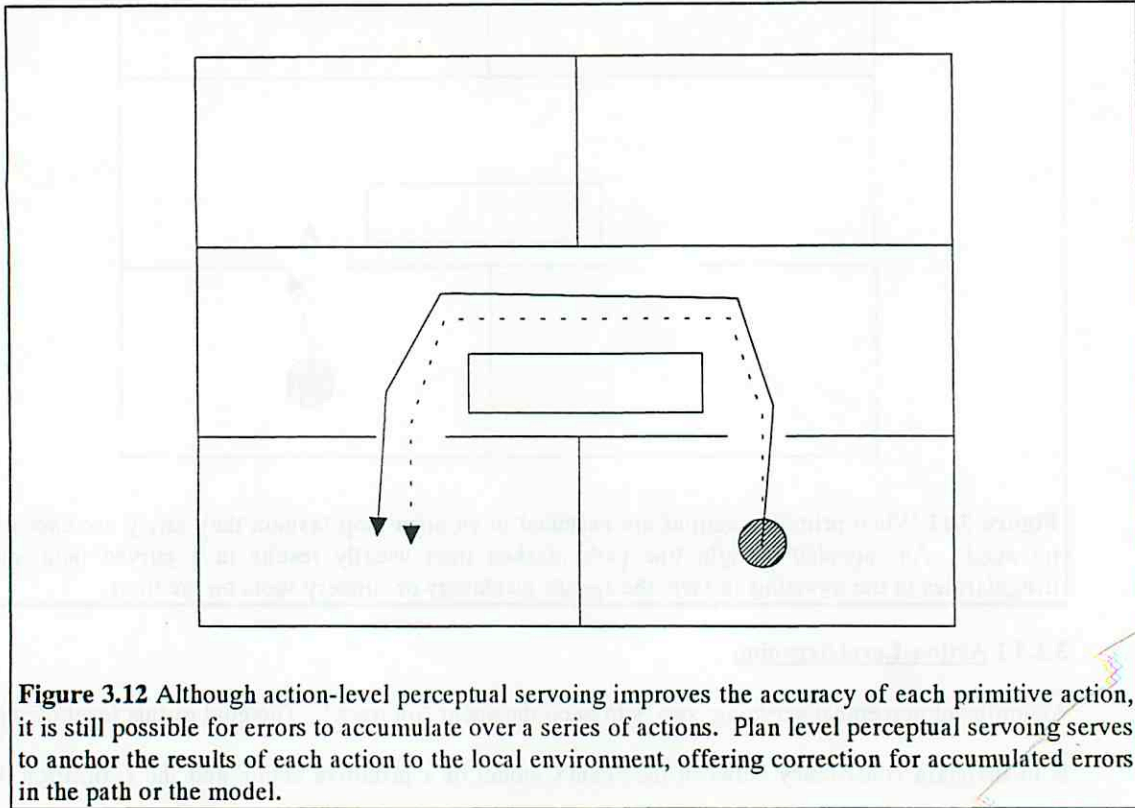
The action-level servo loop can be summarized by the following algorithm:

Action-Level Perceptual Servoing

1. Reason a little:
 - a) Select a landmark from the model which is suitable for steering.
 - b) Predict the agent pose in Δt
 - c) Predict the location of and appearance of the landmark in the image in Δt
2. Move a little

Wait Δt and acquire a new image.
3. Look a little
 - a) Locate the image of the landmark using the prediction of its location to focus the search.
 - b) If the image of the landmark is as expected then do nothing
else determine necessary change in heading and make a corrective action
4. If the primitive action is complete then quit else go to 1.

This algorithm is discussed in more detail in chapter 6 along with results showing that it is possible for action-level perceptual servoing to maintain very precise paths.



3.2.3.2 Plan-Level Servoing

Plan-level perceptual servoing serves to keep the agent "on plan". The goal of this level of servoing is to maintain consistency between reason (as reflected in the plan sketch) and reality (the accumulated effect of the primitive actions). Even though individual primitive actions are accurately executed it is possible for errors to accumulate and for models to be globally inaccurate (Figure 3.12).

Plan-level servoing uses the milestones to determine whether or not the accumulated effect of the primitive actions already executed have had the desired effect. The reasoning system predicts what landmarks should be visible and how they should relate to the agent. Plan level perceptual servoing can be outlined as follows:

Plan-Level Perceptual Servoing

1. Reason a little:
 - a) Predict the location which should result from achieving the next primitive subgoal.
 - b) Select landmarks which should be visible from that location
 - c) Predict the location of the landmarks in the image at that location
2. Move a little
Perform the primitive actions which should satisfy the primitive subgoal.
3. Look a little:
 - a) Locate the image of the landmarks
 - b) If the image of the landmarks are as expected then do nothing
else make corrective adjustments in the plan-sketch
4. If the plan-sketch is complete then quit, otherwise go to 1.

Plan level servoing is discussed in more detail in Chapter 6. The results there show that it is possible to make the possible plan adjustments.

3.2.3.3 Goal-Level Servoing

Goal-level perceptual servoing serves to keep the agent "aiming for the goal". The goal of this servoing loop is to maintain consistency between the plan sketch and the goal. Plans can be incorrect, the model can be incorrect, and it is possible for the agent to make a gross error (Figure 3.13).

Goal-Level: Remember the Goal

1. Reason a little
Construct a plan sketch.
2. Move a little
Perform the primitive actions which would satisfy the first subgoal.
3. Look a little
 - a) Locate the image of the landmarks
 - b) If the image of the landmarks are absent then determine agent's location.
4. Redefine goal in terms of the newly determined location.
5. If the goal is satisfied then quit, otherwise go to 1.

Goal level servoing has not been implemented, but is discussed at some length in Chapter 6.

3.3 Outline of Chapters 4, 5 and 6

The next three chapters describe the details of the experimental system. Chapter 4 describes the representations used in the experimental system and indicates how the "percept" is implemented. Chapter 5 discusses plan sketches in more detail, shows how they are constructed, analyzes the complexity of the planning process in the RML paradigm and presents results which illustrate the quality of the plans constructed by the process. Chapter 6 completes the description of the system by describing perceptual servoing, showing results for action and plan level servoing.

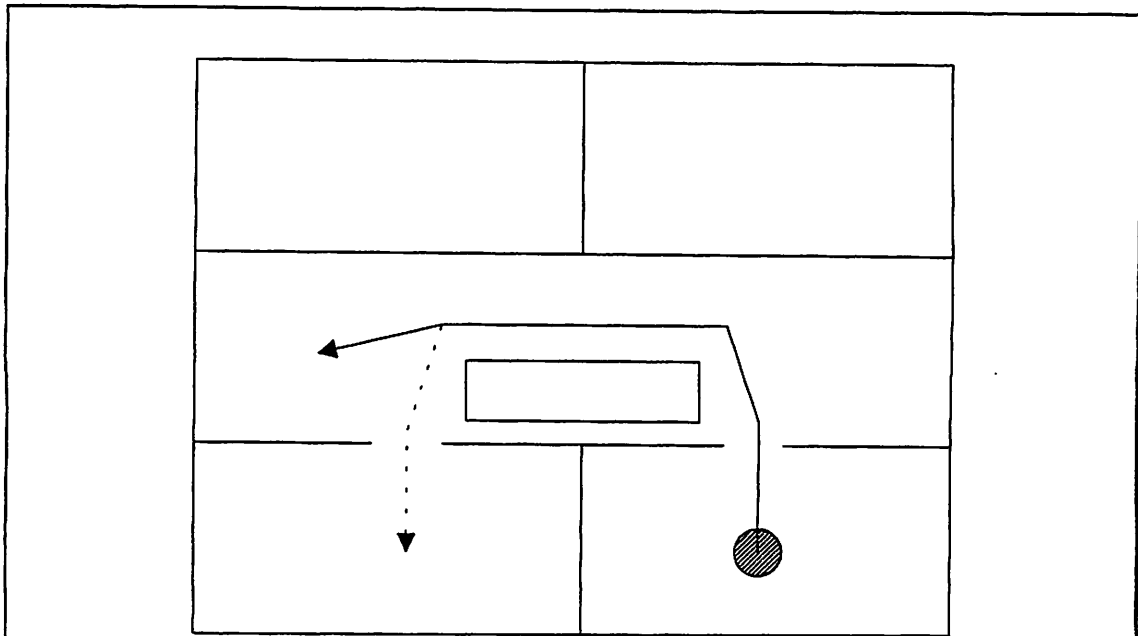


Figure 3.13 Action and plan level servoing work together to control accuracy over complex paths, but if the agent were to encounter a bump, if its actuators were to produce a large erroneous motion or if there were some reasoning error it would be possible for the agent to get lost. In this case action-level and plan-level reasoning will fail and it will be necessary for the agent to relocate itself and readdress the goal.

CHAPTER 4

REPRESENTATIONS

In Chapter 3 an agent was described as developing a "precept": a model its environment, where it is in that environment and what it is currently doing. This chapter describes how that precept is represented in the experimental system. This description is organized into a discussion of three topics:

1. Section 4.1 discusses how the environment and the location of the agent in that environment are represented.
2. Section 4.2 describes the representation of the agent's sensors and effectors. This information is used when verifying that the precept is consistent with "reality".
3. Section 4.3 provides the details of how tasks are represented.

Chapters 5 and 6 show how these representations are used during reasoning, action and perception.

4.1 Representation of the Environment

This environment is usually thought of as consisting of entities referred to as objects, free space, landmarks and light sources. The model described in this section does not make this kind of distinction between various kinds of spatial entities. Rather, it describes the distribution of physical properties in the environment. Spatial entities are described in terms of their shape, the way they interact with light and whether or not they are solid. Spatial entities are also described in terms of their relationships to one another: whether they are connected, whether one is contained in another and whether or not they contain common boundaries. These entities are not, however, classified as objects or landmarks. This model was designed to be used for navigation so particular attention was given to the following representational issues:

1. Physical Properties

Properties of spatial entities such as shape, color, texture, and substance should be represented. As the agent reasons about routes it will need to know which parts of the environment can be traveled through. It will need to know about free space, obstacles, barriers and doorways. When reasoning about perception the agent will need to know about properties which relate to perception, such as shape, reflectivity, transmissivity and specularly.

2. Connectivity

Route planning and reasoning about perception both require that it be easy to determine what entities are near to one another and how, if at all, they connect to one another. When planning a

route through a building, for example, it is useful to know that rooms above one another are "near", but are not directly connected. When reasoning about how to identify a particular 3D landmark it is necessary to be able to determine which patterns are adjacent to a line or a point in 3D. A representation should present these concepts in a computationally efficient way.

3. Localization and local accuracy vs global accuracy.

A spatial representation should offer mechanisms which help the agent cope with imprecision. It is very difficult for an agent to determine its global location with great accuracy. This can have serious consequences. If, for example, an agent were to represent its location in terms of a global coordinate system located somewhere in Chicago, a one percent error in the measurement of its position could place the agent in Northampton, rather than in Amherst. The representational scheme should provide for means of representing this type of error or dealing with it.

4. Scope and perspective

Reasoning about routes and perceptions leads to many processes which are computationally complex. It is essential to be able to focus these processes on relevant information and to be able to ignore irrelevant information. It must be possible to ignore information which is too detailed, too global, too distant or just irrelevant. When reasoning about a path through the UMass research laboratory, for example, we would like to ignore facts about the objects in the top right hand drawer of the desk, facts about the relationship between the earth and Saturn and facts about the streets of San Francisco. A representation of space should offer focusing mechanisms.

The representational scheme described in this section addresses these issues by modelling the environment as a network. The details of this network are described in Section 4.1.1. The network design includes a description formalism which can be used to cope with the problem of localization and accuracy, mentioned above in item 3. This formalism is discussed in Section 4.1.2. The relational mechanisms built into the network set up a system of hierarchies which are used to express scope and perspective (item 4, above). These mechanisms are outlined in subsection 4.1.3. For completeness, the method used for entering model data into the system is described. This description, found in Section 4.1.4, outlines a language used in describing locales and how this language is compiled into the network. Section 4.1 closes with Section 4.1.5, a summary which relates the model to other approaches.

4.1.1 The Network

The environment is represented as a graph or network. The nodes in this graph correspond to frames which represent geometrical entities such as volumes, surfaces, curves and points. Arcs correspond to slots in the frames, describing relationships between the nodes including connectivity, inclusion and structure. This section describes these entities and indicates how they address the issues listed in the introduction to this section.

Unlike the majority of representational techniques¹, which categorize the environment in terms of free space, landmarks and objects, the scheme used here describes the spatial entities of the environment uniformly as subspaces, called locales. A locale can be any volume which has semantic significance to the navigation problem. Each of the following phrases corresponds to a locale:

- "The University of Massachusetts Campus"
- "The Graduate Research Center"
- "The second floor of the Graduate Research Center"
- "Allen Hanson's office."
- "Allen Hanson's desk".
- "The top left hand drawer of Allen Hanson's desk."
- "The eraser in the top left hand drawer of Allen Hanson's Desk."

It is intended that the meaning for the term "locale" capture that found in its common English usage:

"locale \lɔ-kəl\ 1: a place or locality esp. when viewed in relation to a particular event or characteristic. 2: SITE, SCENE <the ~ of a story>." [Webster, 1976]

Through a combination of reasoning and sensing it should be possible for an agent to determine whether or not something, including the agent itself, is in a locale. In this dissertation a locale is defined as a neighborhood N of R^3 , together with a predicate $INSIDE-N(x)$ which is true if and only if x is inside N . A locale system is a partially ordered collection of such neighborhoods. Such a system can be used for localizing the agent by determining the smallest locale, L , such that $INSIDE(agent, L)$ is true.

4.1.1.1 Representing Volumes as Locales

In the network each volume is represented as a locale frame (Figure 4.1). The shape and spatial extent of that volume is represented as a description of its surface. This specifies the R^3 neighborhood the locale represents. To make it possible to describe a locale which contains one or more cavities, the surfaces are represented as a list of connected surface components, beginning with the outside component. Thus the surface pointer in a locale frame would point to a list of the form:

(SC1 SC2 ... SCN)

where SC_1, \dots, SC_N are surface descriptions. A closed, hollow wooden box, for example, could be represented as a locale whose surface description has two components: SC_1 and SC_2 . SC_1 would be a description of the outside surface of the box. SC_2 would be a description of the inside surface, forming

¹ For example see section 2.1 of Chapter 2.

the boundary between the wood and the cavity of the box. The locale, in this case, is the space occupied by the wood the box is made of.

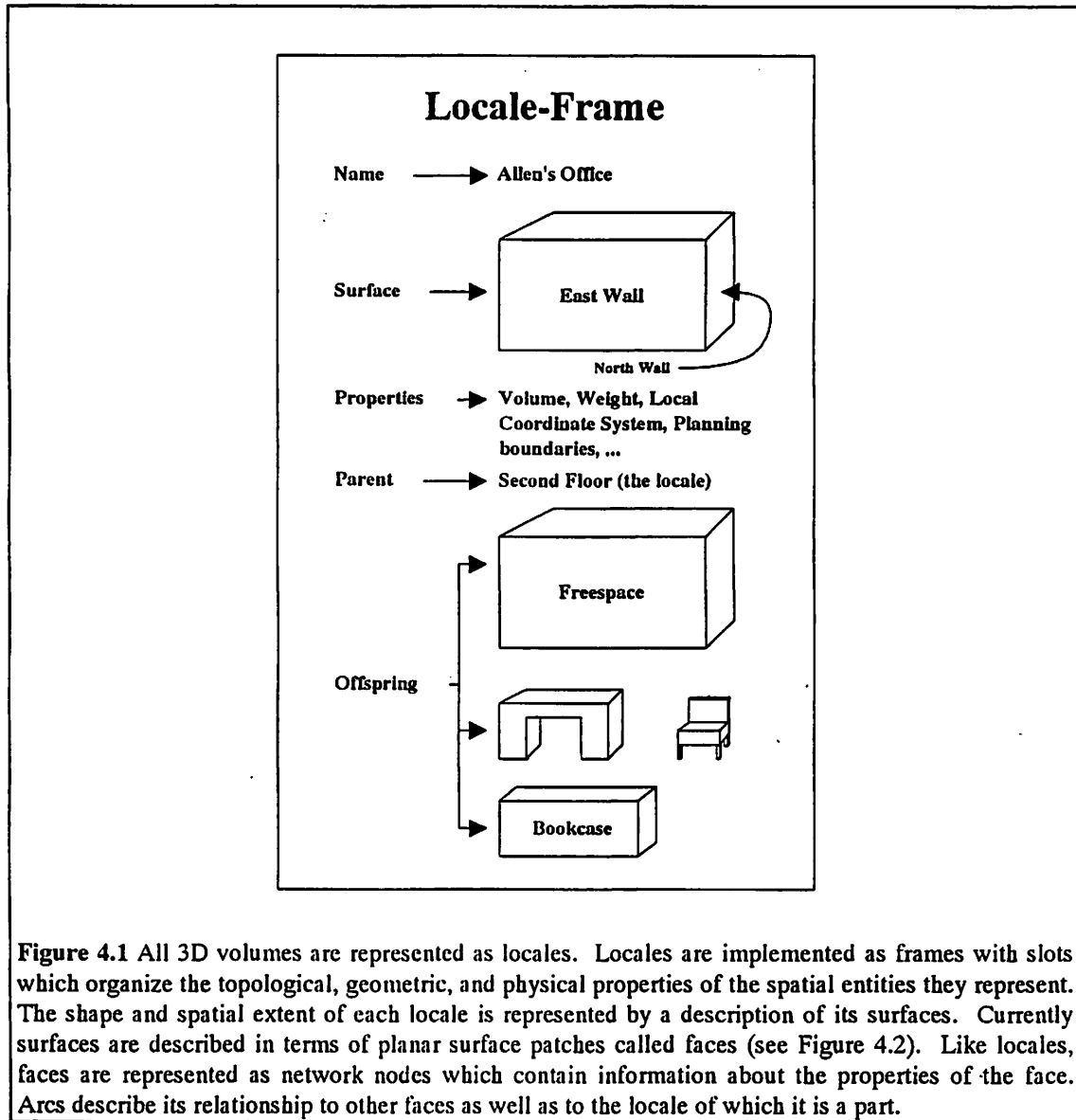


Figure 4.1 All 3D volumes are represented as locales. Locales are implemented as frames with slots which organize the topological, geometric, and physical properties of the spatial entities they represent. The shape and spatial extent of each locale is represented by a description of its surfaces. Currently surfaces are described in terms of planar surface patches called faces (see Figure 4.2). Like locales, faces are represented as network nodes which contain information about the properties of the face. Arcs describe its relationship to other faces as well as to the locale of which it is a part.

Surface components are, in turn, described as a list of surface patches or faces. Each component S_{Ck} of the surface description (SC_1, \dots, SC_N) is represented as a list of the form:

$$(F_1, F_2, \dots, F_m)$$

where each F_j is a face. A face is a portion of the surface of a locale which is homogeneous with respect of some physical property. The property may be related to its composition or its description. In the

experimental system, for example, faces correspond to portions of a plane which are homogeneous with respect to planning-related properties. Faces are planar and are either doors, walls or windows. The representation of faces will be described in more detail later in Section 4.1.1.2.

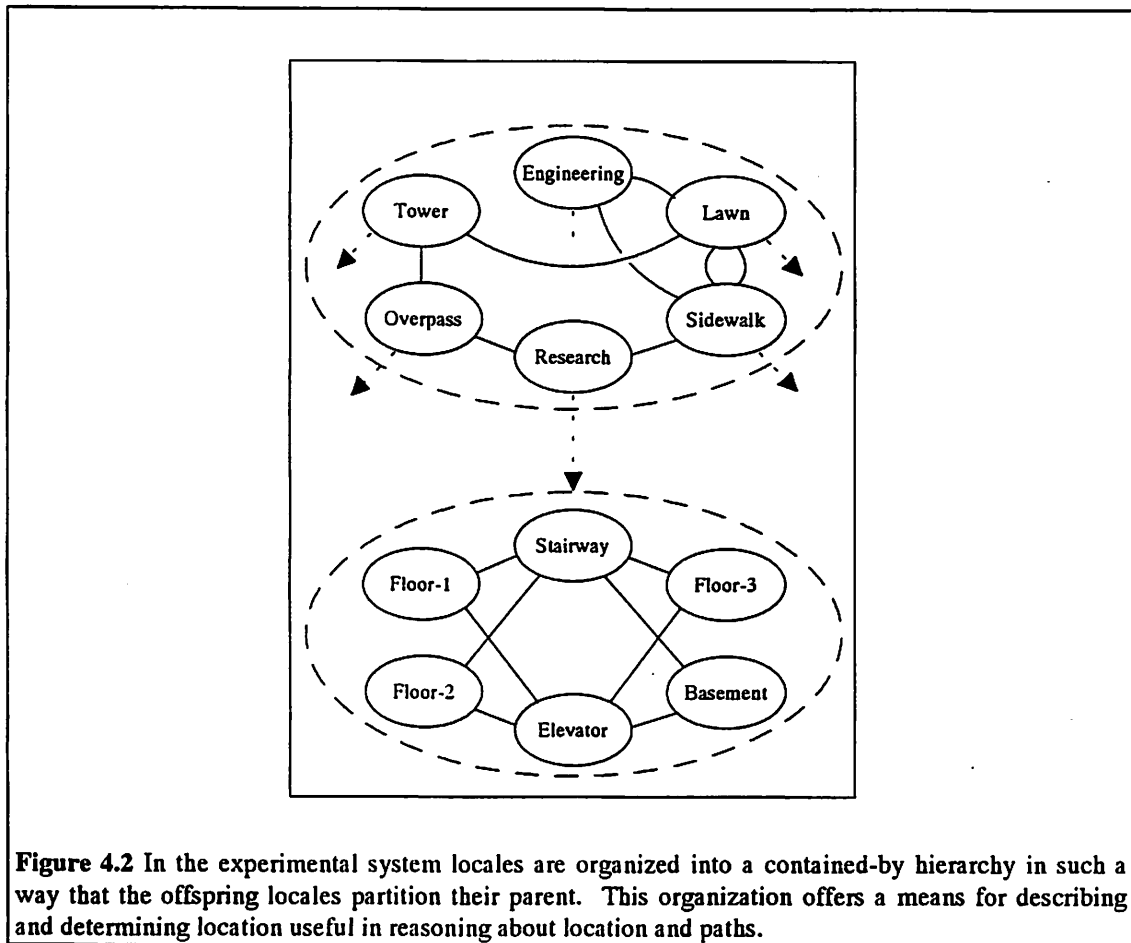
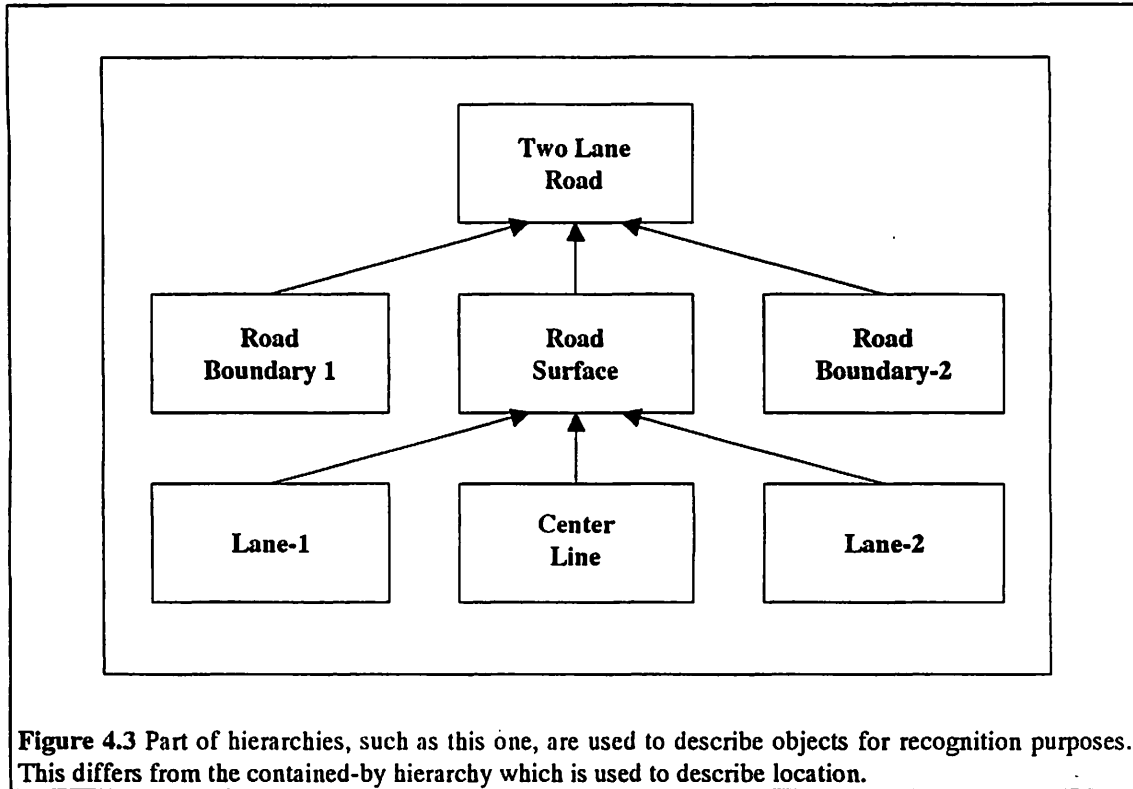


Figure 4.2 In the experimental system locales are organized into a contained-by hierarchy in such a way that the offspring locales partition their parent. This organization offers a means for describing and determining location useful in reasoning about location and paths.

The properties slot of the locale frame points to a property list. This list holds properties associated with the locale, such as volume, weight, local coordinate system and planning information. The most important of these properties is the local coordinate system for the locale. All geometric entities of the locale, its vertices, lines and faces, are described in terms of this local coordinate system. This offers a mechanism for dealing with local and global errors. Once it is known that the agent is inside a locale it can relate itself to the local coordinate system, and thus not be subject of global inaccuracies of the type mentioned earlier. This will be discussed in more detail in Section 4.1.2. A second desirable property of

the fact that locales are described in terms of their own local coordinate system is that when a locale (say a desk) is moved only the position of its local coordinate system needs to be updated.



The parent and offspring slots in the locale frame organize the locales into a contained-by hierarchy (Figure 4.2). Each offspring locale is a subset of its parent locale. As implemented, these offspring locales partition their common parent locale into disjoint subsets. This organization serves a different purpose from what is usually referred to as a part-of hierarchy. Part-of hierarchies are usually used to describe objects as part of an object recognition or image understanding program. Figure 4.3, an example from [Lawton, Levitt, McConnel, Nelson and Glicksman, 1987], shows the description of a two lane road as a part-of hierarchy. The descendants of each node in the figure are subsets of their parents here too, but this set theoretic statement is not what is emphasized. What is emphasized is that there is a recognizable object, called a two lane road, which has recognizable parts, called borders and road surface. The part-of hierarchy has been used either to partition the recognition process or to suggest further recognition processing based on a partial recognition. In the contained-by hierarchy, on the other

hand, the stress is on the set theoretic notion of "contained-by". The division of space described by the hierarchy is used as a means for describing the location of a point and for describing the scope of reasoning activities.

Location descriptions take advantage of the locale hierarchy. Since each offspring of a locale is smaller than its parent and encompasses a different space from any of its siblings, making the statement that something is in an offspring locale is a stronger locational statement than stating that thing is in the parent locale. Section 4.1.2 describes the way location is defined in the experimental system and outlines the advantages of this representation. The localizing property offered by this hierarchy is also useful in focusing the reasoning processes involved in planning and perception. Section 4.1.3 discusses these mechanisms.

4.1.1.2 Faces

As has already been mentioned, faces are used in surface descriptions. Like locales, they are represented as frames (Figure 4.4). The boundary slot of these frames point to a description of the shape of the face. In the experiments described in this dissertation faces were described as planar polygons. To allow for the specification of faces which contain holes, these boundaries have been described as a list of connected boundary components:

(BC1 BC2 ... BCN)

Each BC_k in this list is a boundary component represented as a list of the line segments which make up the closed curve of that component. In this list BC₁ is the outermost boundary component. BC₂, ... , BC_N are boundary components corresponding to holes in the face. A wall with a window in it would be represented as a face with a two component boundary. The outer component would describe the extent of the wall; the inner component would describe the boundary between the wall and the window.

The properties slot points to a property list, containing information about the physical properties of the face. These properties include area, longest boundary line, a two dimensional local coordinate system and certain planning information used by the plan sketching mechanism described in Chapter 5. Properties which relate to the perception of a face are also kept on its property list. These properties, which may vary over the surface of a face, are described as a list of regions. Regions are defined as

patches of the face which are uniform with respect to perceptually related properties. For vision these properties include reflectance, specularity and transmissivity. Because perception of the inside of a face may differ from the perception of its outside, faces have two sets of regions: an inside set and an outside set.

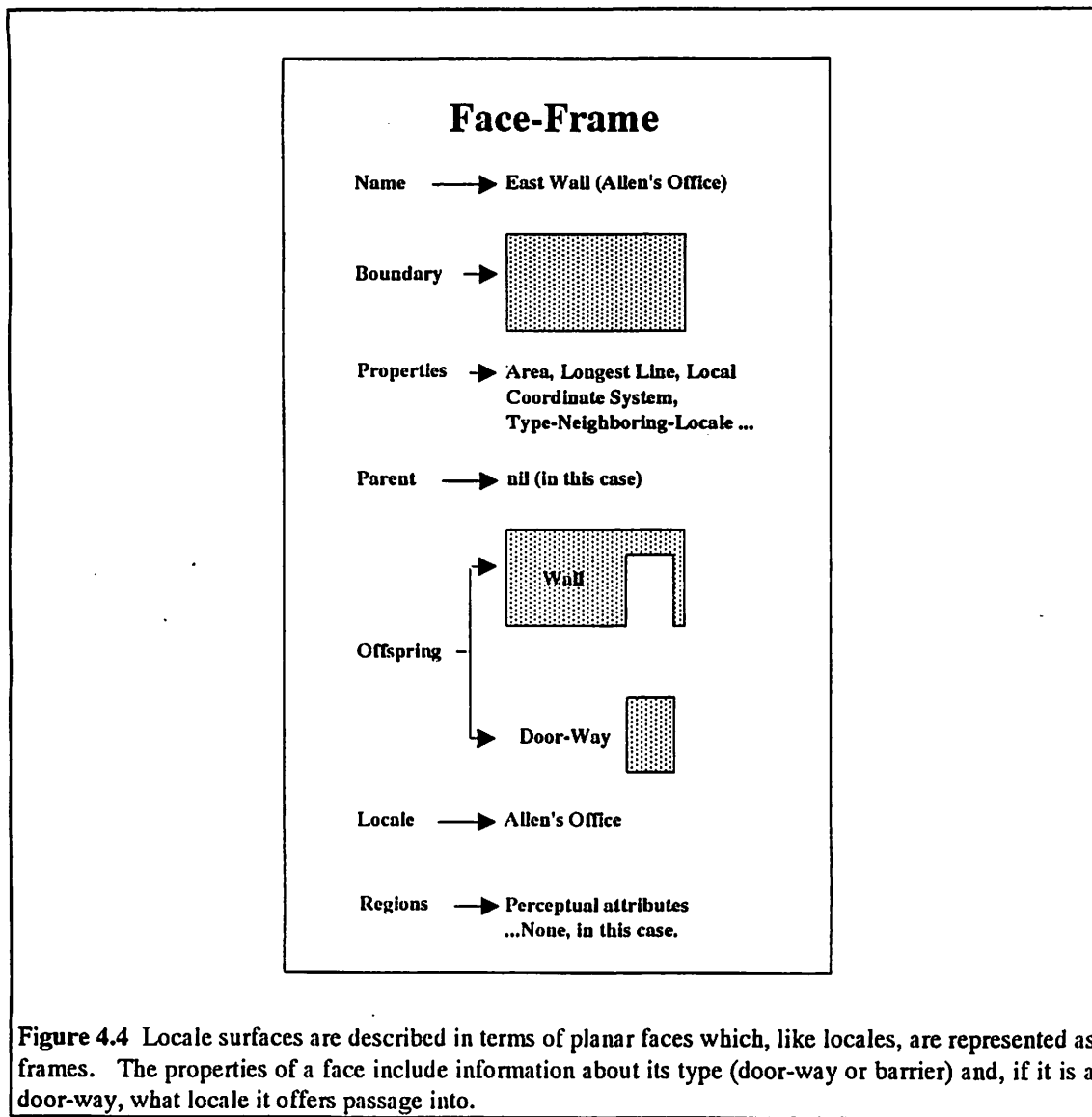


Figure 4.4 Locale surfaces are described in terms of planar faces which, like locales, are represented as frames. The properties of a face include information about its type (door-way or barrier) and, if it is a door-way, what locale it offers passage into.

Faces, like locales, are organized into a contained-by hierarchy (Figure 4.5). This organization is used when reasoning about free paths and visibility. Each of these processes amounts to determining whether or not a straight line intersects a face (or subface) with particular properties. In the case of free path determination the task is to determine if a proposed line of travel intersects a face with the property

"barrier". When checking for visibility the task is to determine whether or not a line of sight intersects a plane which is opaque. Intersection searches are done by first checking each face which corresponds to a root node in one of these hierarchies. If an intersection with a face is found, its offspring faces are checked. The search continues down the hierarchy until the leaf node face is found. The property of this subsurface determines whether or not the intersection blocks the path (or visibility). If the branching factor of this hierarchy is β and it is δ levels deep then a search of all the leaf nodes would be of complexity β^δ . Organizing the search the way just described reduces this complexity to $\beta\delta$. The locale network of the experimental environment has an average β of about 6 and a δ of at least 3. This means that only 18 faces are considered instead of 216. Savings can be considerable even in this relatively simple environment.

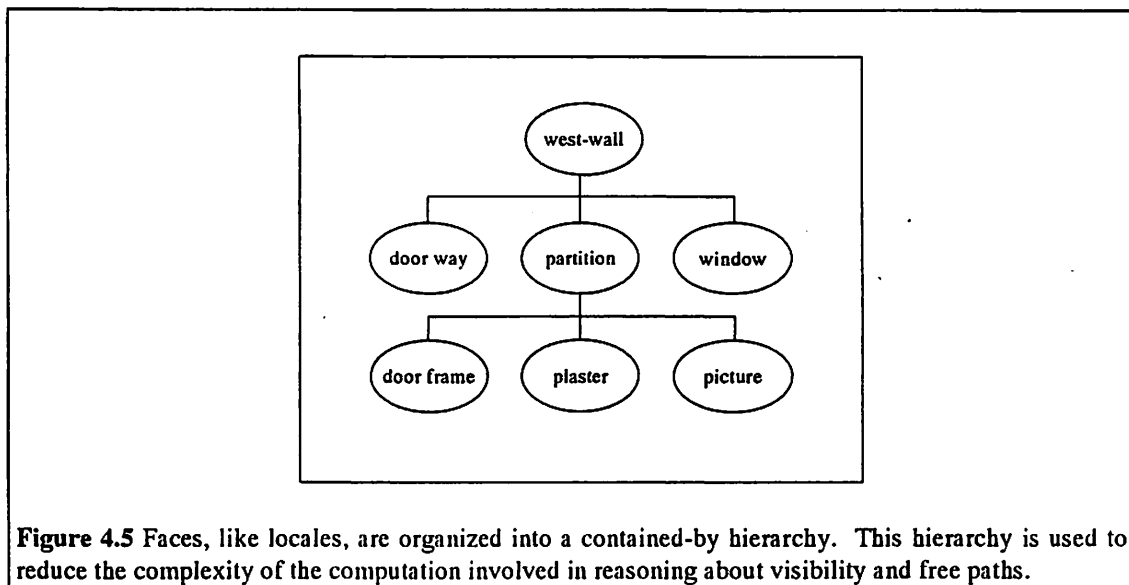
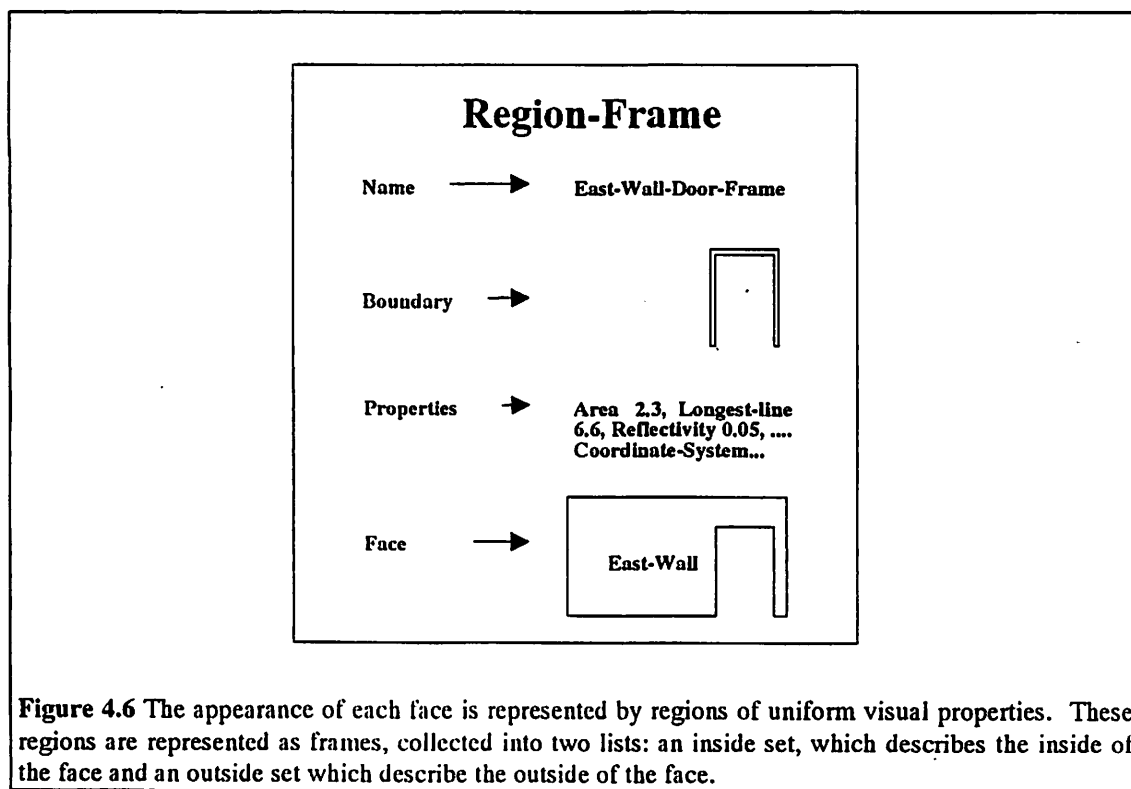


Figure 4.5 Faces, like locales, are organized into a contained-by hierarchy. This hierarchy is used to reduce the complexity of the computation involved in reasoning about visibility and free paths.

4.1.1.3 Regions

Regions represent patches on a surface which are uniform with respect to perceptually related properties. The use of the term "region" in this context is not accidental. The construct is intended to be an internalization of the region concept originally described by [Brice and Fennema, 1970]. As interpreted in that work, regions were uniform areas of the image plane. Here the concept has been moved "into" the model.

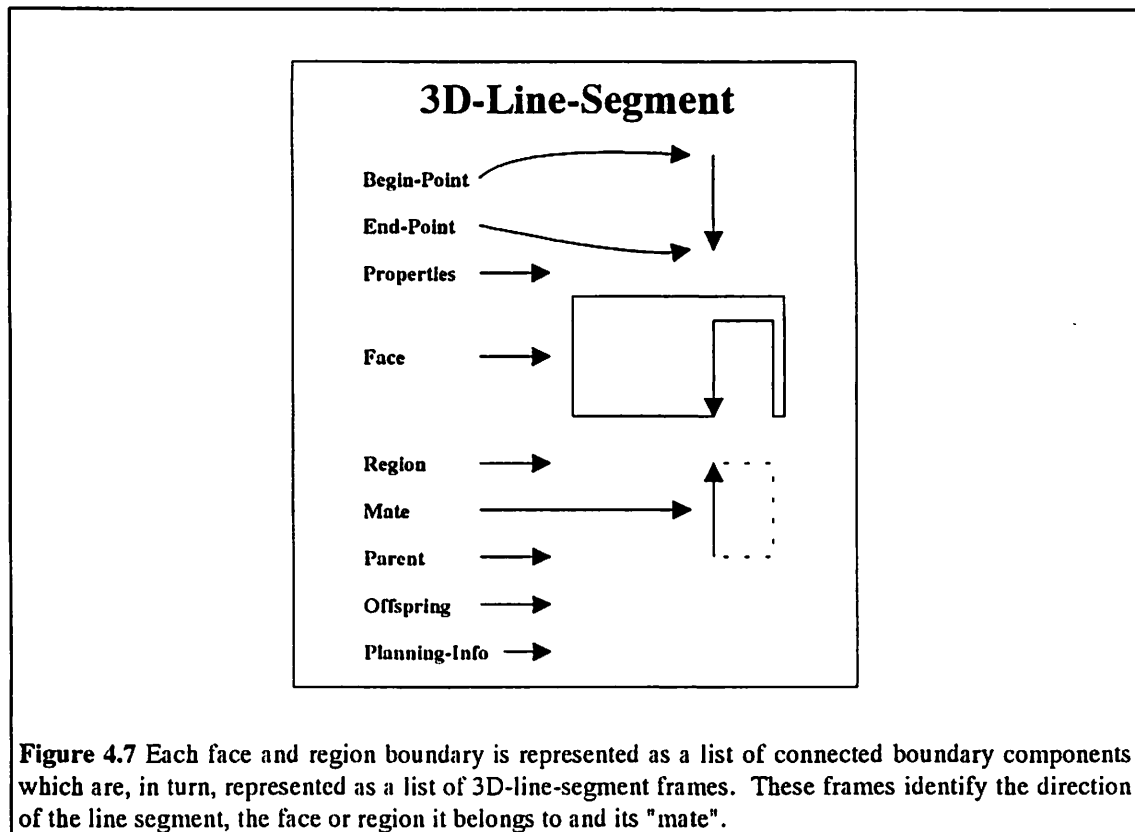
Regions are represented in the network as region-frames. These frames are similar to face-frames (compare Figures 4.4 and 4.6). The principle difference between the two is the way they are used. As with faces, they are planar polygonal patches, whose shape is described by the content of the boundary slot. Their property slot points to a property list, itemizing the properties of the region, including area, local coordinate system, longest line and perceptual properties. The perceptual properties of a region could include reflectivity, specularity, transmittance, texture or color. In the experiments described in the following chapters only reflectance information was used. The face slot points to the face of which it is a part. Unlike faces, regions have no offspring slots. They can be thought of as the leaf nodes of the face hierarchy.



4.1.1.4 Line Segments

Line segments are used primarily to describe boundaries. Like the other nodes in the network, they are implemented as frames (Figure 4.7). Line segments are directed and as such have "begin-points" and "end-points" as indicated in Figure 4.7 by the similarly named slots. Line segments are used in face and region boundary descriptions in such a way that, when viewed from outside the locale, a region or face is

to the left of each of its boundary line segments. If the line segment is part of a region boundary then the "region" slot contains a pointer to the appropriate region frame. This provides access to the perceptual properties associated with one side of the line segment. The perceptual properties of the other side of the line are obtained in a similar way using the line segment pointed to by the "mate" slot. The "mate" is a line segment belonging to the boundary of region which shares this boundary segment. If a line segment is not part of a region boundary it may be part of a face boundary, identified by the "face" slot in exactly the same way. A line segment may not be part of the boundary of a region and a face at the same time. A 3D line can be represented by several 3D-line-segment frames, each capturing the fact that it is part of a different boundary curve.



As with locales and faces, line segments are part of a hierarchy. This hierarchy is induced by the other hierarchies. As a face is partitioned, its boundary segments are partitioned. As each line segment is partitioned the results are stored in its offspring slot, forming a line-segment hierarchy. This hierarchy is a representational convenience, but it has not been used in this work.

The planning techniques described in Chapter 5 plan locally in two dimensions. Locales are constructed in such a way that each has a planar floor. Areas with very uneven floors are broken into sublocales which have approximately planar floors. While global planning is done in three dimensions, local planning is done in two, based on the floorplan of the locale. For this purpose planning boundaries are constructed and used to summarize information about barriers and doorways. This information is stored in a line segment planning-info slot. The information in this slot is of the form:

(planning-category *type* neighboring-locale *locale*)

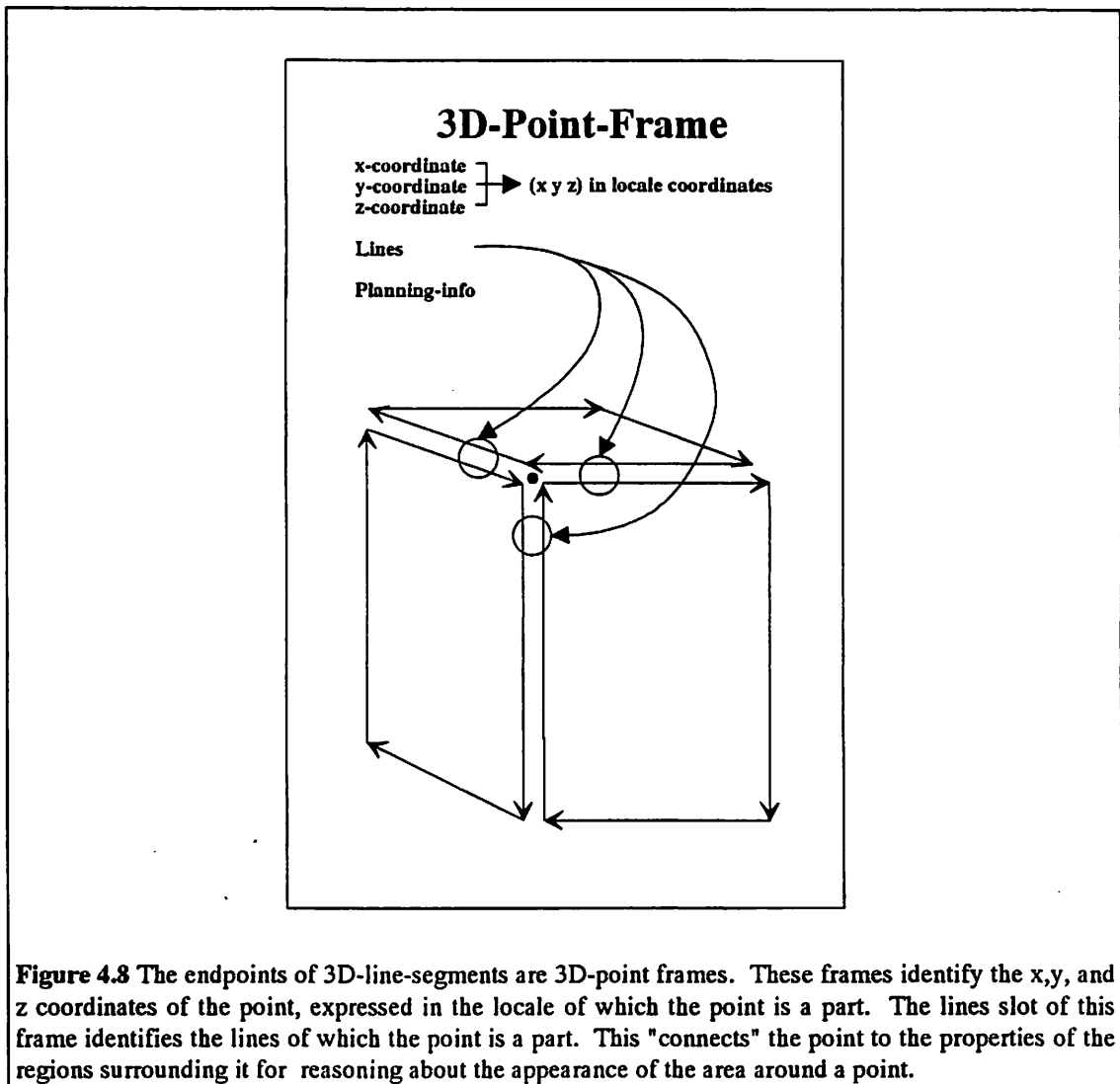
where *type* is either "barrier", meaning that the agent cannot pass through that segment, or "door-way", meaning that it can. In either case *locale* identifies the neighboring locale, if it is known. This is discussed more in Chapter 5.

4.1.1.5 Points

Points are used to represent vertices and locations of special interest. Each point is represented by a "3D-point-frame" (Figure 4.8). Each point is associated with a locale and its x, y and z coordinates are expressed in terms of the coordinate system of that locale. Points which are part of a boundary are associated with the line segments of which they are a part. These line segments are identified in a list pointed to by the "lines" slot of the frame. As shown in Figure 4.8, a point can be part of the boundary of several faces and regions. This information is used to access the properties of the faces and/or regions surrounding the point. The perceptual servoing techniques described in Chapter 6 use this to determine the reflectance of the regions surrounding a point so the point can be identified an image.

Some points are used to mark nodes in the planning space described in Chapter 5. These points contain planning information. In particular a point marking a door-way exit from one locale will contain the identity of the neighboring locale if it is known. This information is stored in the planning-info slot.

The locales, faces, regions, 3D-line-segments, and 3D-points are the building blocks of the network. Using them, the network describes the shape and physical properties of space in such a way that the properties and the space they describe are "computationally close" to each other. This addresses the issues 1 and 2 identified in the introductory part of Section 4.1. The issue of localization and accuracy is discussed next, in section 4.1.2. This will be followed by a discussion of scope and perspective in 4.1.3.



4.1.2 Localization and local accuracy vs global accuracy

As was mentioned in point 3 in the discussion at the beginning of section 4.1, one of the objectives set out for the locale structure was to deal with the problem of localization: how to determine where the agent and other spatial entities are located in the environment. There are three specific navigation related problems in this regard:

1. When moving from place to place the agent will at times need to determine its location. As it steers its way through a doorway it may have to determine this position quite accurately in order to make the proper adjustments to its course so it can avoid causing damage, both to itself and to the environment.

2. When it has completed a plan subgoal it will need to be able to determine whether or not the goal has been successfully completed. This may involve determining location with different degrees of accuracy.

3. When the agent first "wakes up" or it somehow "gets lost" it will need a strategy for determining where it is. This is the "Where am I?" problem mentioned in Chapter 4 (in particular, see Figure 1.2).

These problems all involve the notion of location or place. This notion of location is different from the (x,y,z) coordinates in some global coordinate system. The inability to make measurements arbitrarily accurately limits our ability to use a global coordinate system for detailed navigation, especially when trying to navigate through tight spaces. To cope with this problem a location is defined in terms of a locale and a pose. The locale in this definition offers a local coordinate system and the pose describes a position (and, if desired, an orientation) with respect to that coordinate system. A location is represented as a frame (Figure 4.9).

location-frame	
locale	-----> A locale frame
pose	-----> The coordinates and orientation of the place, expressed in the coordinate system of the locale

Figure 4.9 A place is represented by a location, which is a frame identifying a locale containing the place and the coordinates of the place as expressed in the local coordinate system of that locale.

Both the locale and the pose serve to locate the place. The locale specifies an R^3 neighborhood containing the point. This localizes the point to a degree. Humans use this degree of localization when they use English expressions like:

- ".. in Amherst"
- "... on the third floor".
- "...in the office"
- ".. in the desk drawer."

These expressions are inexact when viewed from a local perspective, since two distinct points can be in the same locale, as shown in Figure 4.10 (a). However from a global perspective, as in Figure 4.10 (b) they can be fairly exact statements. The locational ambiguity of the phrase "... in the desk drawer" is very small when viewed from the perspective of the Massachusetts locale. When navigating within a locale, however, knowing the locale offers little locational help. It is for this reason that the pose information is included. The pose is also a frame, as shown in Figure 4.11.

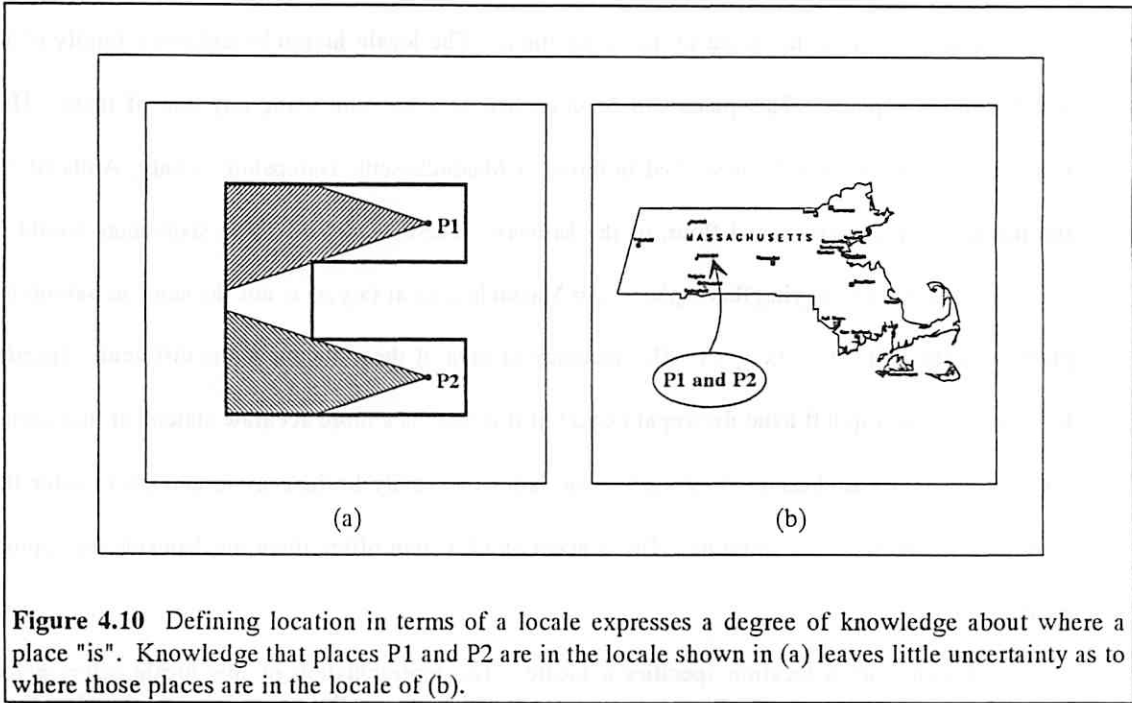
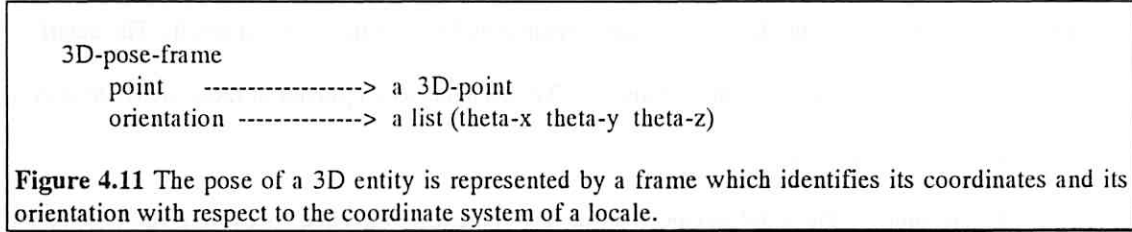


Figure 4.10 Defining location in terms of a locale expresses a degree of knowledge about where a place "is". Knowledge that places P1 and P2 are in the locale shown in (a) leaves little uncertainty as to where those places are in the locale of (b).

The point slot in this frame is a 3D-point and the orientation slot identifies a list of three angles. The convention used here is that the orientation is that which results by applying rotations in the following order: theta-z, theta-y and then theta-x. When the orientation information is irrelevant it is set to nil. This would be the case when identifying the pose of something which has on orientation information, such as a sphere or a point. When the entity being located has a description in terms of its own coordinate system then the pose is used to locate that coordinate system. When specifying the location of the agent, for example, the agent coordinate system is specified as having its origin at the center of the agent's footprint and situated so that the x-axis points in the direction the agent is facing, the y-axis pointing to its left and the z-axis pointing upwards. The location of this agent is specified by a locale and the pose of the agent's coordinate system with respect to the locales local coordinate system.



The same place can be described by many locations. The locale hierarchy defines a family of locales which contain a place. This place can be described as a location using any one of them. Harvey's position can, for example be described in terms of Massachusetts, Hampshire county, Amherst, UMass the research center, the second floor, or the hallway. The accuracy of these statements would not be equivalent, however. Saying that a place is in Massachusetts at (x,y,z) is not the same as saying that the place is in the hallway at (x',y',z') . The accuracy in each of these statements is different. Specifying a location as in the top left hand drawer at (x,y,z) , if it is true, is a more accurate statement than saying that the location is Massachusetts at (x',y',z') . The latter can easily be inaccurate enough to infer that the location is in the incorrect building. The concept of a location offers three mechanisms for coping with localization:

1. **Recognition:** A location specifies a locale. The representation of this locale offers perceptual information which can be used to determine whether or not the agent is in the locale. Once the locale is verified, the perceptual information can be used to determine the pose.
2. **Neighborhood:** Knowledge of being in a locale makes a definite statement about locality in terms of a neighborhood (Figure 4.10). Knowledge about the locale and the distance of the agent from the center of its coordinate system can also be used to estimate the uncertainty of the pose as expressed in that coordinate system.
3. **Relationship:** A location specifies, via its locale, the relationship of a place to the locales that contain it and are near to it. These relationships can be used to guide a well contained reasoning and perceptual process to use in the "Where am I?" process. This is described more in Chapter 6.

These mechanisms are used extensively in the reasoning and perceptual processes described in the following chapters.

4.1.3 Scope and Perspective: The Locale Hierarchy

As mentioned in point 4 of section 4.1 it is important to be able to focus reasoning. The exponential nature of reasoning methods, such as the A* algorithm used in plan sketching (Chapter 5) and most perceptual reasoning methods, can be made tolerable by keeping the problem small. The detailed effects of this will be described in Chapters 5 and 6. The mechanisms employed in these areas are what we will call perspective and scope.

The meaning of the word perspective, in this context, is intended to capture the common English usage of the word expressed in definition 2b of the following passage from the dictionary:

"³perspective n [MF, prob. modif. of OIt *prospettiva*, fr. *prospetto* view, prospect, fr. L *prospectus* - more at PROSPECT] 1 a : the technique or process of representing on a plan or curved surface : LINEAR PERSPECTIVE b : the technique of adjusting the apparent sources of sounds (as on a radio program) into a natural-seeming and integrated whole 2 a : the interrelation in which a subject or its parts are mentally viewed : CONFIGURATION b : the capacity to view things in their true relations or relative importance <by failing to maintain a historical ~ they ... fail to see the solution -- Herbert Ratner> ..." [Webster 1977]

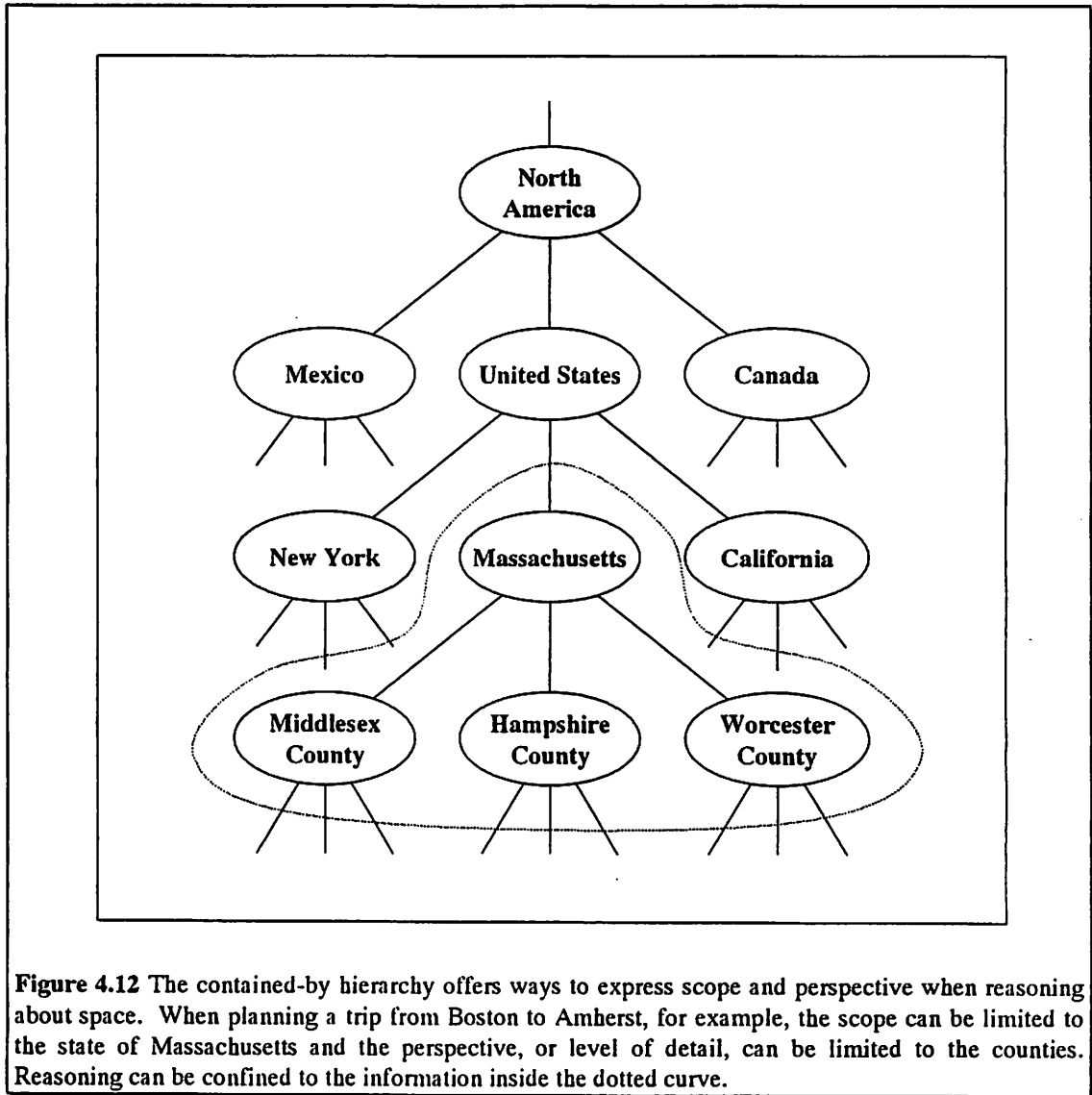


Figure 4.12 The contained-by hierarchy offers ways to express scope and perspective when reasoning about space. When planning a trip from Boston to Amherst, for example, the scope can be limited to the state of Massachusetts and the perspective, or level of detail, can be limited to the counties. Reasoning can be confined to the information inside the dotted curve.

When reasoning about a trip from Boston to Northampton, for example, the distinction between P1 and P2 in Figure 4.10 (b) is unimportant. The concept of location described in Section 4.1.2 offers a mechanism which can be used to express this perspective difference. When reasoning about the trip, the two locales "Boston" and "Amherst" define location. By knowing that we are in Boston and want to go

to Amherst we can reason about ways to leave Boston and to get into Amherst and not worry about the details of either. We may be able to limit reasoning to the information available to the county level (see Figure 4.12).

Scope, in this context, refers to the range of operation. When planning our trip from Boston to Amherst it is important to be able to ignore information about New York, Europe and Saturn. This additional information would be irrelevant to the trip and would add complexity to the planning effort. In Figure 4.12 this is accomplished by limiting attention to Massachusetts. Scope and perspective are used to considerable advantage by the planning mechanism described in Chapter 5.

4.1.4 Compiler

The environmental model specifies the geometric and perceptual structure of the robot's world. Portions of the interior of the University of Massachusetts Graduate Research Center, as well as a portion of the surrounding campus, were modelled and embedded in the locale network. All measurements were taken by hand using a tape measure. A coordinate system was chosen for each locale and its features were measured from its origin. The measurements were converted into statements in a model description language and these statements were compiled into network entities. This section offers a brief overview of the language.

Locales are described in three steps. The first step describes the overall structure of the locale. An example of this type of description is shown in Figure 4.13. This description, written in COMMONLISP, defines the locale in terms of some carefully measured points p1, p2, ... , p16. The measurements are converted into properly initialized "3D-point" frames (Section 4.1.1.5) by the create-3d-point function as shown in that Figure. The locale is then constructed using the make-locale function, in this case giving it the name south-west-lobby. The surface is described as a list of components, each consisting of a list of planar faces, built by the "create-face" function from the points p1, p2, ... , p16. Finally "enter-locale" then enters the locale into the network as an offspring of the locale called "second-floor" whose path from the top of the locale hierarchy is (environment lgrc second-floor). With respect to the second-floor locale coordinate system, the south-west-lobby locale coordinate system is located at (77.55, 29.70, 0.00) and has an orientation of (0.0 0.0 0.0).

```

(describe-locale
  (enter-locale
    (let
      ((origin (create-3D-point '(0.00 0.00 0.00)))
       (p1 (create-3D-point '(2.19 0.00 0.00)))
       (p2 (create-3D-point '(2.19 0.00 8.46)))
       (p3 (create-3D-point '(2.19 0.71 0.00)))
       (p4 (create-3D-point '(2.19 0.71 8.46)))
       (p5 (create-3D-point '(0.00 0.71 0.00)))
       (p6 (create-3D-point '(0.00 0.71 8.46)))
       (p7 (create-3D-point '(0.00 8.21 0.00)))
       (p8 (create-3D-point '(0.00 8.21 8.46)))
       (p9 (create-3D-point '(21.98 8.21 0.00)))
       (p10 (create-3D-point '(21.98 8.21 8.46)))
       (p11 (create-3D-point '(21.98 0.00 0.00)))
       (p12 (create-3D-point '(21.98 0.00 8.46)))
       (p13 (create-3D-point '(21.26 0.00 0.00)))
       (p14 (create-3D-point '(21.26 0.00 8.46)))
       (p15 (create-3D-point '(18.14 0.00 0.00)))
       (p16 (create-3D-point '(18.14 0.00 8.46))))
      (make-locale
        :name 'south-west-lobby
        :surface
          (list (list
            (create-face 'pillar-south-face (list (list p3 p1 p2 p4)) nil)
            (create-face 'pillar-east-face (list (list p5 p3 p4 p6)) nil)
            (create-face 'north-doorway (list (list p7 p5 p6 p8))
              '(type (door-way (environment lgrc second-floor lobby))))
            (create-face 'east-wall (list (list p9 p7 p8 p10)) nil)
            (create-face 'south-wall (list (list p11 p9 p10 p12)) nil)
            (create-face 'south-west-wall (list (list p13 p11 p12 p14)) nil)
            (create-face 'west-doorway (list (list p15 p13 p14 p16)) nil)
            (create-face 'north-west-wall (list (list p1 p15 p16 p2)) nil)
            (create-face 'floor (list (list p9 p11 p13 p15 p1 p3 p5 p7)) nil)
            (create-face 'ceiling (list (list p8 p6 p4 p2 p16 p14 p12 p10)) nil)))
          :parent nil
          :offspring nil))
      '((environment lgrc second-floor) ((77.55 29.70 0.00) (0.00 0.00 0.00))))

```

Figure 4.13 The first step in describing a locale is to specify the overall structure of the locale and its relationship to its parent. The expression in this figure specifies the locale named "south-west-lobby" in terms of planar faces such as the ones named "pillar-south-face", "pillar-east-face" and "north-doorway". The relationship between this locale and its parent is specified in the last line of the expression. Its parent, the second floor of the research building, is specified by the path (environment lgrc second-floor) and its location and orientation are given as (77.55 29.70 0.00) and (0.00 0.00 0.00), respectively.

Once the overall shape of the locale has been defined, a detailed structural description is specified as illustrated in Figure 4.14. In this figure one of the faces in the surface description of the locale identified by the locale hierarchy path (environment lgrc second-floor south-west-lobby) is being partitioned into

wall and doorway. That face, identified as the (south-wall), is partitioned by this expression into two subfaces, one called wall and one called door-way. In the face hierarchy of the south wall these will become known by their path descriptions (south-wall wall) and (south-wall door-way). Each is described as to its type (wall or door-way) and its boundary, described in terms of the local two dimensional coordinate system of the south-wall.

```
(partition-face
  ((environment lgrc second-floor south-west-lobby)
   (south-wall))
  ((wall
    (type (wall))
    (((0.00 0.00)(2.00 0.00)(2.00 6.88)(6.92 6.88)(6.92 0.00)(8.21 0.00)(8.21 8.46)
      (0.00 8.46))))))
  (door-way
    (type (door-way (environment lgrc second-floor hallway-north-end)))
    (((2.00 0.00) (6.92 0.00) (6.92 6.88) (2.00 6.88))))))
```

Figure 4.14 Once the overall structure of a locale has been defined, details can be specified by partitioning the faces into smaller subfaces. Here the face named "south-wall" is partitioned into a subface named "wall", which is a barrier to travel (type (wall)), and a subface named "doorway", which offers passage to the locale named "hallway-north-end".

When it is known that a face is shared by another locale, the path description of the other locale is entered as part of the type expression. In Figure 4.14 the door-way face is shared by the locale called "hallway-north-end". The path for that locale is given by the underlined portion of the expression:

```
(door-way
  (type (door-way (environment lgrc second-floor hallway-north-end)))
  (((2.00 0.00) (6.92 0.00) (6.92 6.88) (2.00 6.88))))))
```

After the structural descriptions are complete the visual descriptions are entered using an expression like the one in Figure 4.15. This describes the inside of the wall portion of the face partitioned in Figure 4.14 as consisting of three regions: one called the door-jam, which has reflectance 0.07; one called the wallboard, which has reflectance 0.31; and one called the baseboard which has reflectance 0.07. The coordinate pairs in this description represent the boundaries of the region, described in the two dimensional coordinate system of the face (south-wall wall).

4.1.5 Summary and Comparison

The environmental model in the experimental system is constructed as a network, called the locale network. The nodes in this network are frames which represent spatial entities (volumes, faces, lines and points) and their physical properties (eg. reflectivity, barrier vs. non-barrier). The arcs capture the relationships that exist between these spatial entities (eg. containment and contact). The locale network describes the environment as a distribution of physical properties. Kuipers et al. [Kuipers, 1978], [Kuipers, 1982] and [Kuipers and Byun, 1988a, 1988b], also describe the environment as a network, but the nodes in their network represent significant "places", such as Harvard Square or MIT (Figure 4.17a). An arc between two nodes is used to represent a path between those two "places". Their network contains no details concerning these paths. They view an agent as having path following and a place recognition capabilities. Representation of the paths is embedded in the path following algorithms.

```
(describe-face
  'inside-appearance
  '((environment lgrc second-floor south-west-lobby)
    (south-wall wall))
  '((door-jam
    (reflectance 0.07)
    (((0.00 0.00) (0.21 0.00) (0.21 0.50) (0.21 8.46) (0.00 8.46))))
    (wallboard
    (reflectance 0.31)
    (((0.21 0.50) (0.72 0.50) (0.72 8.46) (0.21 8.46))))
    (baseboard
    (reflectance 0.07)
    (((0.21 0.00) (0.72 0.00) (0.72 0.50) (0.21 0.50))))))
```

Figure 4.15 The appearance of a face is described by breaking it into regions of uniform appearance. This expression specifies the appearance of the "wall" portion of the "south-wall" as consisting of three appearance regions, named "door-jam", "wallboard" and "baseboard".

It is from descriptions of this type that the network is constructed. The "compiler", which bears the name "describe-locale", uses these descriptions to create the frames, the local coordinate systems, the property lists, the hierarchies and the planning information used in Chapter 5. Figure 4.16 shows a perspective view of the hallway-north-end locale, constructed from the network. This figure shows the kind of detail present in the model.

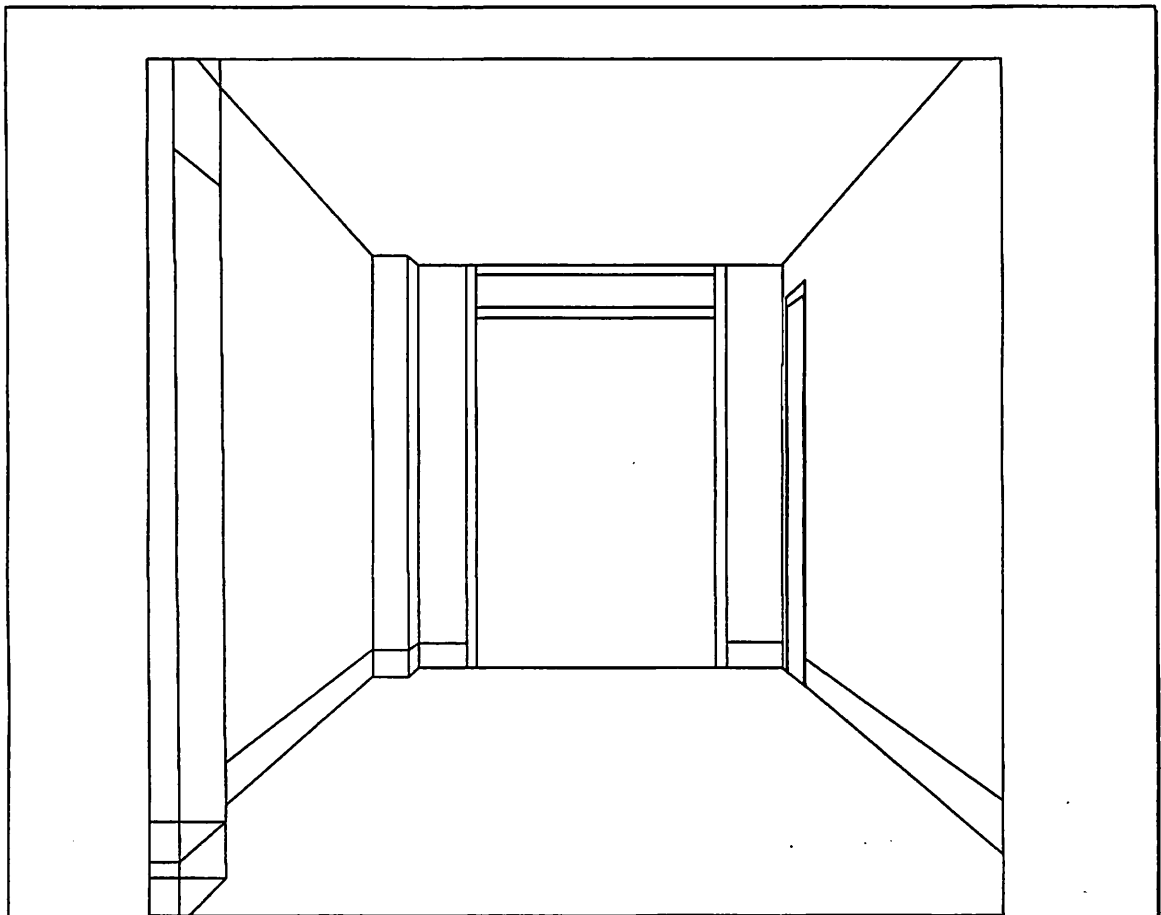


Figure 4.16 The model of the environment is incomplete, but locally positionally accurate. All objects are modeled but many details may be absent. This scene, which is projected from the model, shows walls, baseboards, moulding and pillars. Details, like wall sockets and posters, are not always in the model.

The representation of space in the locale network, on the other hand, is centralized. All properties of space and places used by the agent are in the network. The concept of a "place" is expressed as a "location", as was described in Section 4.1.2. Locations can be used to represent any point in the environment and they provide access to all the known physical properties of the portion of the environment near the places they represent. This feature is used extensively by both the reasoning and perceptual processes as described in Chapters 5 and 6.

The arcs in the locale network capture a number of relationships, but the most important one is that of containment. The contained-by relationship maintained in the locale, face and line hierarchies results in a spatial partitioning which, as we have seen in Section 4.1.2, is useful for describing and reasoning

about location. It is also a powerful tool for describing scope and perspective, as was discussed in Section 4.1.3. Kuipers briefly described what he called "regions" [Kuipers 1978] as a collection of nodes (Figure 4.17b). He described the use of regions to represent a collection of places, such as a city. His regions described two dimensional areas and were not organized into a hierarchy. Although they did contain nodes or "places", *containment* was not the region property which was used in his work. Locales, on the other hand, represent three dimensional volumes and are organized into a hierarchy which specifically uses the containment property for describing places, scope and perspective.

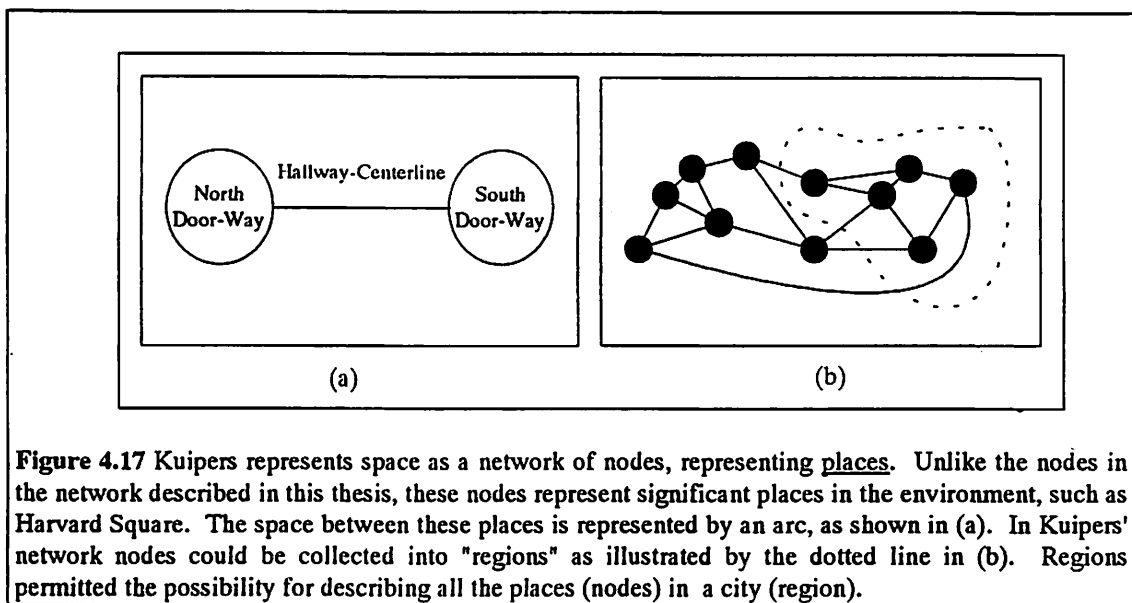


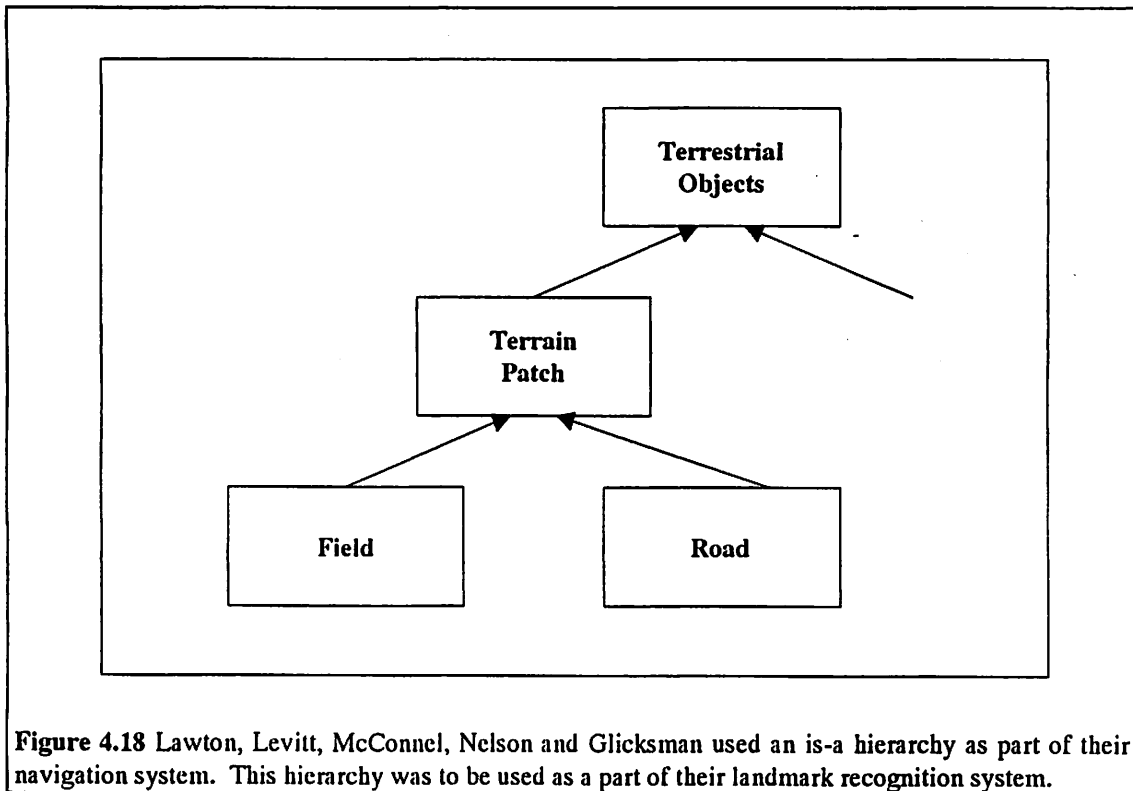
Figure 4.17 Kuipers represents space as a network of nodes, representing places. Unlike the nodes in the network described in this thesis, these nodes represent significant places in the environment, such as Harvard Square. The space between these places is represented by an arc, as shown in (a). In Kuipers' network nodes could be collected into "regions" as illustrated by the dotted line in (b). Regions permitted the possibility for describing all the places (nodes) in a city (region).

The point was made in Section 4.1.1.1 that the "contained-by hierarchy" used in the locale hierarchy is also different from the often discussed "part-of" hierarchy. Part-of hierarchies have been used primarily to describe the decomposition of objects as part of a perception process, rather than a locational process. The use of a part-of hierarchy is based on the *classification* of each node in the hierarchy as a type of object, not on the amount of space occupied by each node in relationship to its parent or child.

No attempt has been made in the locale model to attach "cultural" categories to spatial entities. The knowledge that a locale is a desk or a laboratory does not seem to be required for navigation. No object or landmark recognition of this type is done by the experimental system. For this reason concepts such as the part-of (Figure 4.3) or the is-a (Figure 4.18) hierarchies used by [Lawton, Levitt, McConnel, Nelson

and Glicksman, 1987] are not implemented in this system, although their inclusion would be a simple matter. Cultural categories seem to be required for other purposes, such as conveying information to a traveler about an area he knows nothing about or requesting that the agent retrieve a specific object. A discussion of these points is, however, beyond the scope of this dissertation.

The environmental model described in this section is quite different from any of those described in Chapter 2. It is significantly more structured. This structure addresses one of the significant problems with those systems: computational complexity. The locale hierarchy provides focusing mechanisms which are used in the reasoning processes described in Chapters 5 and 6 to reduce the complexity of the reasoning and perceptual processes significantly. This claim is supported in those chapters both by mathematical arguments and experimental results.



4.2 The actuator and sensor models

In order to verify that reasoning and reality are consistent it is necessary to have a model of the interface between the agent and its environment. Without this model a reasoning process can neither

predict the consequences of activating an actuator nor interpret the input from the sensors. This interface has been modeled in terms of an actuator model and a sensor model.

Primitive Action	Action Model
(agent-move d) heading = h	<pre> If (distance ≤ 2 ft.) then new-location-frame locale ←-- old-location--frame-locale pose ←-- pose-frame point ←-- old-point + d*(cos (h), sin(h)) orientation ←-- (0 0 h) else nil. </pre>
(agent-turn angle)	<pre> If (angle ≤ 180 degrees) then new-location-frame locale ←-- old-location--frame-locale pose ←-- pose-frame point ←-- old-point orientation ←-- (0 0 h+angle) else nil. </pre>

Figure 4.19 For Harvey the actuator model is relatively simple. For small distances $d \leq 2$ feet Harvey is modeled as moving in a straight line. For greater distances the prediction is undefined. This means that large movements must be decomposed into a series of steps of no more than two feet. The translational error due to turning an angle less than 360 degrees is sufficiently small that it can be predicted as happening as intended. Larger turns are unlikely, but the model is undefined.

The actuator model describes movement in terms of primitive actions. Primitive actions have already been described in Chapter 3 as one of a finite number of basic movements which, taken together, can describe all the possible actions the agent is capable of performing. In the actuator model each of these primitive actions is identified and associated with an action model which predicts the consequences of taking that primitive action. Harvey has been modeled as having two primitive actions:

1. (agent-move d) - moving a distance d along a "straight line".
2. (agent-turn theta) - turning about his axis an angle theta.

As simple as these primitive actions are, the realities of the environment and the construction of the agent make it unlikely that their action models can be accurately expressed for large movements. In Harvey's

case, for example, the flexibility of the tires and the unevenness of the traveling surfaces cause "straight-line" motion to be significantly curved and "rotation in place" to result several inches of unintended translation. Fortunately, the translational motion introduced by rotations of less than 180 degrees is less than an inch and, for forward motions of a foot or two, the path traveled can be modeled as if Harvey were following a straight line. Harvey's actuator model is summarized in Figure 4.19. This model is implicitly represented in the implementation of the action-level perceptual servoing code.

The sensor model identifies each of the sensors and associates with each of them an interpretation model. The interpretation model is a model of the sensor as an interface to the environment and is intended to be free of "semantics". It is an interpretation of how the sensor data came into existence. The data from the blind man's cane discussed in Chapter 1, for example, is only a force. To interpret this force, however, it is necessary to have a model for the cane and how it is being used.

Harvey is equipped with a TV camera and a ring of sonar sensors, but only the TV camera was used in the work described here. The interpretation model for this TV camera specifies that the data in an image array is the result of an optical system consisting of a lens which has projected an image of the environment onto a planar surface into which is imbedded a photosensitive array of finite extent. The lens is modeled, according to the principles of geometric optics, as transforming the scene according to a perspective transformation. This transformation is determined by the focal length of the lens and the location of the camera. This model is implicit in the procedures used to reason about what should be sensed when checking to see that "reality" and "reason" are consistent. This reasoning is described in Chapter 6.

When reasoning about the appearance of the environment the projection of a locale onto the sensor array is represented in terms of structures which are two dimensional equivalents of the those used for representing the environment. Two dimensional entities are described in terms of frames representing 2D-points and 2D-line-segments. When the agent reasons about what it might see from a particular location it creates portions of what it might see using information in the locale network. Structures associated with the locale identified by the location frame are transformed into camera coordinates by a function called "transform-3D-to-camera-coordinates" and then projected as 2D-points and 2D-line-

segments onto the image plane using a function called "project-3D-to-2D". This function performs projection onto the two dimensional plane, performing three dimensional clipping as it does so.

The photosensitive array embedded in this two dimensional plane is represented as an $m \times n$ array of raster-structures. These structures, like those in 3D and 2D, are represented as frames called raster-points and raster-lines. The 2D structures are mapped into these structures by the transform "2D-to-raster" which converts the 2D coordinates into row and column coordinates. This conversion is done according to the number of array elements per foot in the x and y directions and the 2D-coordinates of the center of the array. Two dimensional clipping is performed at this stage.

4.3 Representing tasks and the current state of the problem

Tasks for this system are navigation tasks. They are represented goals. As the system reasons about a goal it decomposes it into a set of subgoals and keeps track of the current state of the problem by organizing these subgoals into what is called a plan sketch. These plan sketches contain subgoals yet to be accomplished and milestones which are used to determine whether or not those subgoals have been successfully accomplished.

4.3.1 Goals

Goals are represented using conceptual dependency notation [Schank and Abelson, 1977]. Two types of goals are represented. The first type is a goal to move from one location to another. These are represented as ptrans expressions of the form:

(ptrans location-1 location-2).

This expresses the goal of moving form location-1 to location-2.

The second type of goal is used to represent a change in point of view or perspective. This happens whenever the agent passes from one locale to another. This type of goal is represented using Schank's notion of a "mental change" and is denoted

(mtrans location-1 location-2)

In this situation location-1 and location-2 are expected to represent the same "place", but from different perspectives. When passing through a door, for example, there is a point where the agent considers itself to be "outside" the room it is leaving and "inside" the other. This mental change is modeled by an mtrans

expression. All mtrans goals are primitive. They are achieved by transforming the pose as specified in local coordinate system of the locale associated with location-1 to that of the locale associated with location-2.

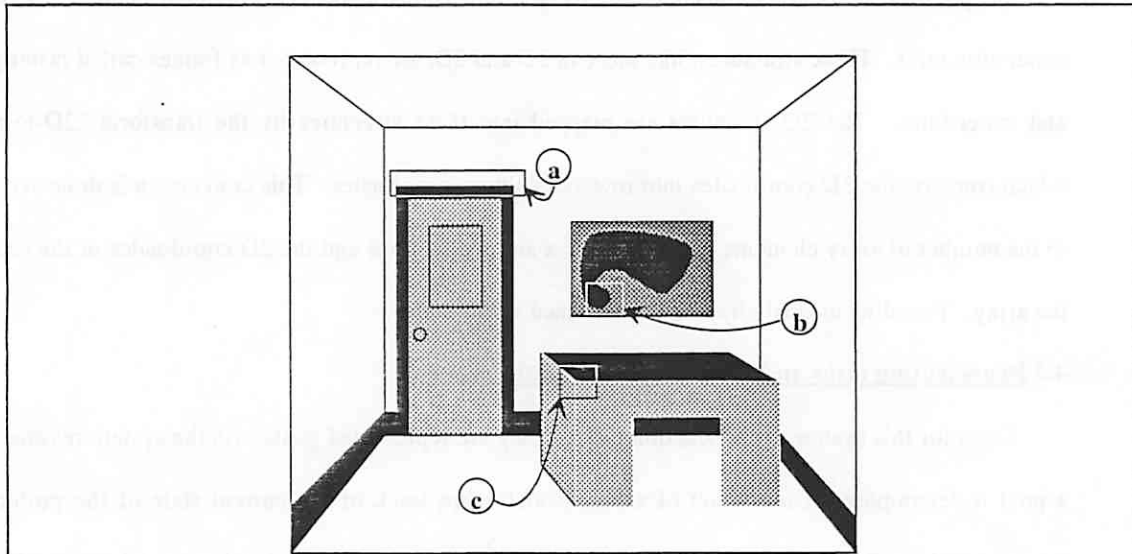


Figure 4.20 Landmarks are 3D structures which have physical properties which make them distinct from other 3D structures nearby and which are locally unique.

4.3.2 Milestones

The concept of a milestone was briefly described in Chapter 3. They are one of the constructs used for determining whether or not reason and reality are consistent. When a milestone is recognized the agent's view of where it is in a plan is consistent with reality. Milestones are used to determine whether or not a goal or subgoal has been achieved. Given the goal

(ptrans robot-lab allen's-office)

a milestone which might be used to verify that this goal has been accomplished might be:

"You should see a large black and brown desk to your right".

In the experimental system, milestones are described in terms of 3D landmarks, which may be any distinctive 3D-structure, such as the ones indicated in Figure 4.20. The term distinctive means easily distinguished from other nearby 3D-structures, using the available sensors. A milestone is a set of landmarks, together with a description of their expected spatial relationship with respect to the agent.

These milestones are used to determine that the goal has been completed (Figure 4.21). Milestones are represented as a location. A location specifies a locale and a pose. The locale specifies the physical nature of the local environment and the pose specifies what portion of the locale can be seen.

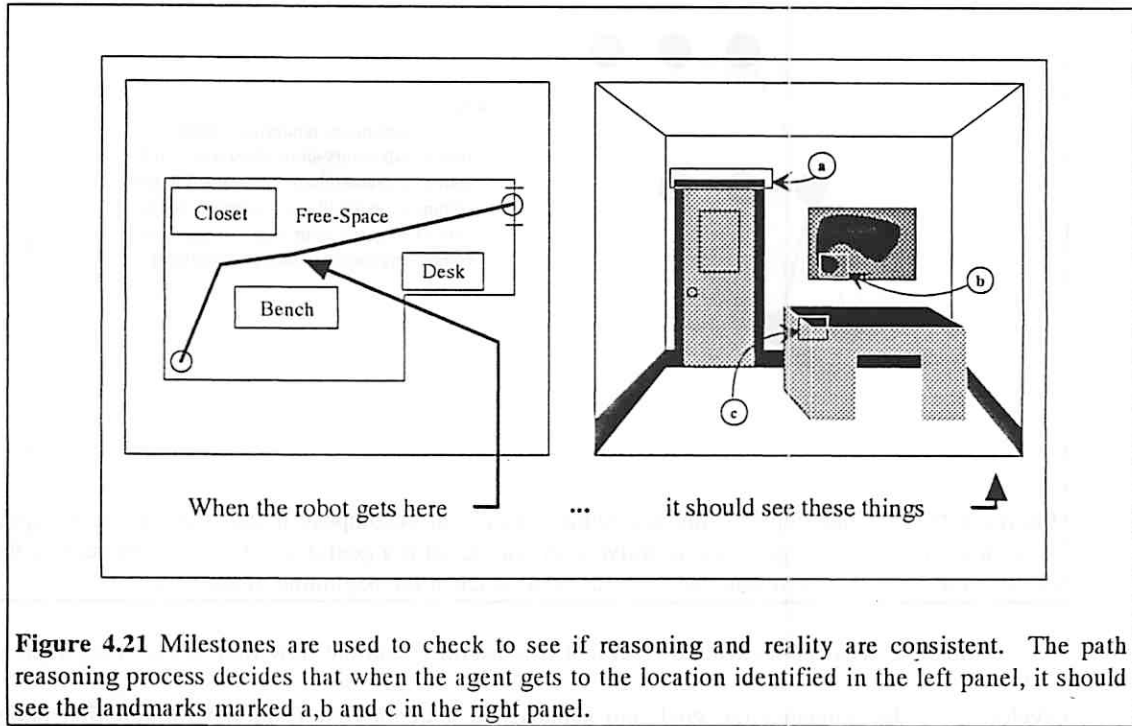


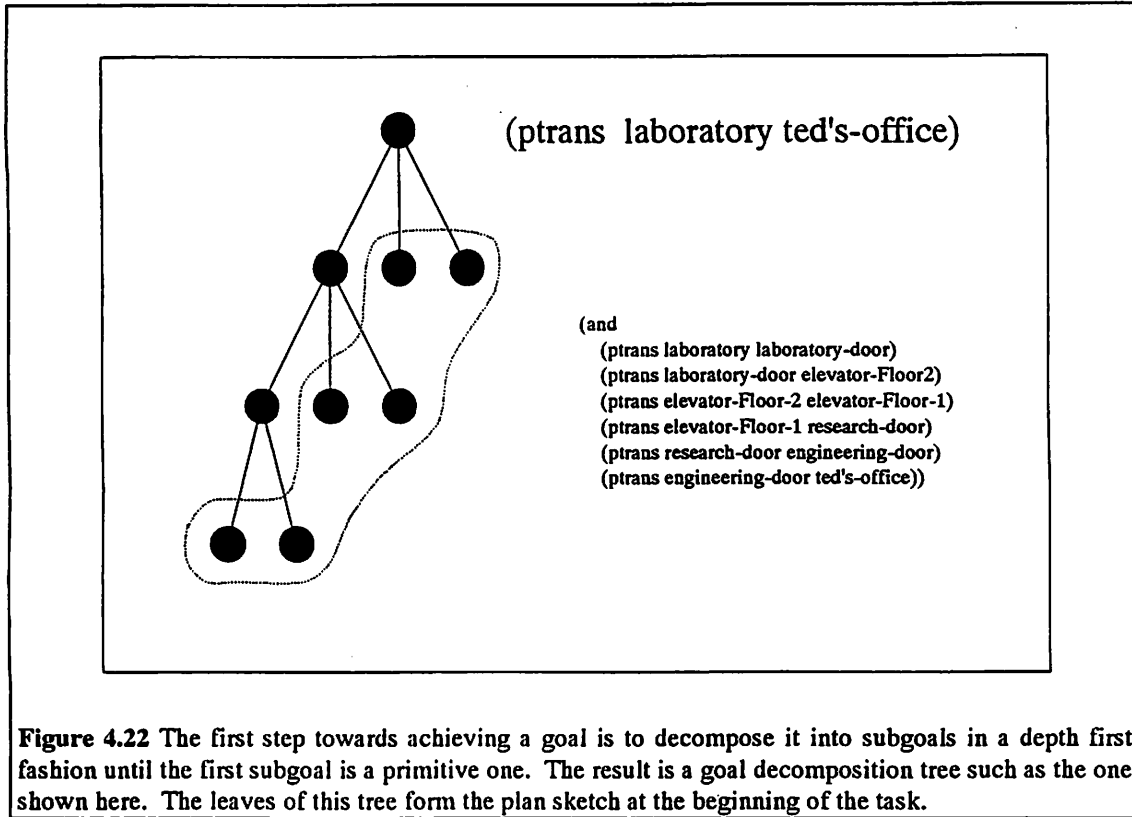
Figure 4.21 Milestones are used to check to see if reasoning and reality are consistent. The path reasoning process decides that when the agent gets to the location identified in the left panel, it should see the landmarks marked a,b and c in the right panel.

4.3.3 Plan Sketches

The current state of the task is represented as a plan sketch. The plan sketch represents what is left to be done to achieve an overall goal and specifies the milestones to be used in checking progress with reality. A plan sketch would be a list the form:

(M0 (ptrans a b) M1 (ptrans b c) ... (ptrans f g) Mg)

In this list M0 is a precondition milestone which must be satisfied before the plan sketch can be considered meaningful. If the milestone M0 is recognized then it is appropriate to attempt the first subgoal (ptrans a b). If (ptrans a b) has been satisfied, then M1 should be recognized and it is appropriate to undertake the next subgoal. Failure to recognize a milestone means that some the plan sketch must be modified. The name "plan sketch" is used for these lists because they represent rough "plans" and modifications to them are expected to happen often.



To illustrate what plan sketches represent it is useful to outline how they arise. Plan sketches are developed by decomposing the goal into subgoals in a depth first fashion until the first subgoal is primitive (Figure 4.22). A primitive subgoal, as defined in Chapter 3, is one which can be directly executed without further subgoal refinement. Given the primitive actions of Section 4.2, a subgoal of the form:

(ptrans location-1 location-2)

would be primitive if the path from location-1 to location-2 were unobstructed. This subgoal could ideally be satisfied by executing the script:

```
(script-ptrans-clearpath (location-1 location-2)
  (turn (- (angle location-1 location-2)) heading)
  (move (distance location-1 location-2)))
```

This script specifies that to get from location-1 to location-2 it is only necessary to perform two actions: first turn the difference between the angle of the line from location-1 to location-2 and the current robot

heading, then move the distance between these two locations. As we shall see in Chapter 6 this script is oversimplified, but it serves here to illustrate the point.

When a primitive goal is accomplished it is removed from the sketch and the next subgoal in the plan sketch is refined as before, resulting in a new tree similar to what is shown in Figure 4.23. Plan sketches change often, but usually only near their beginning. Later steps tend to be abstract and are less likely to be incorrect for that reason.

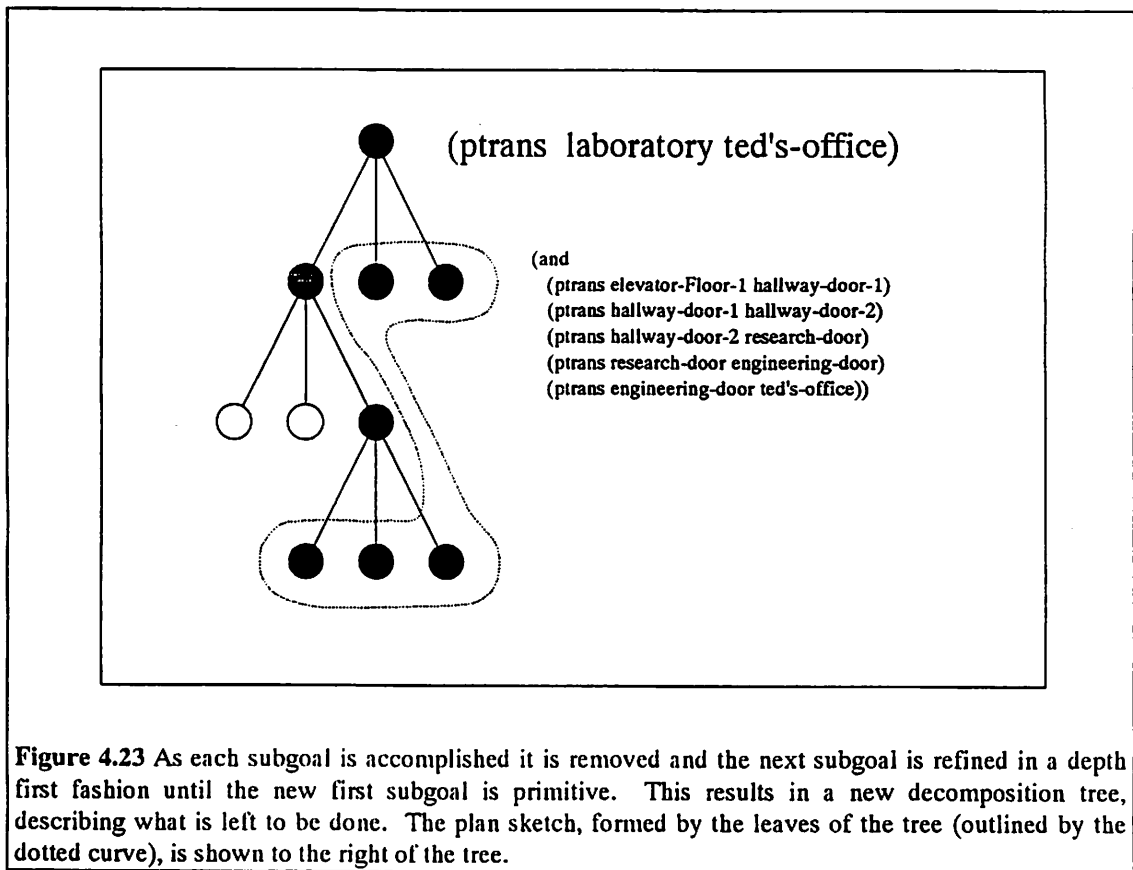


Figure 4.23 As each subgoal is accomplished it is removed and the next subgoal is refined in a depth first fashion until the new first subgoal is primitive. This results in a new decomposition tree, describing what is left to be done. The plan sketch, formed by the leaves of the tree (outlined by the dotted curve), is shown to the right of the tree.

CHAPTER 5

SKETCHING PLANS

The plan-and-monitor algorithm described in Chapter 3 develops plans incrementally using a process referred to as "plan sketching". This chapter describes an algorithm which uses the scope and perspective mechanisms mentioned in Chapter 4 to implement a version of that process and examines the complexity and quality of the planning component of the resulting system.

Together, sketch-a-plan and plan-and-monitor develop 3D plans hierarchically according to levels of abstraction offered by the geometric description of the environment. The result is a dramatic improvement in planning efficiency over an algorithm like A* as presented in [Nilsson, 1980]. Under the assumption that no replanning is necessary, both analytical and experimental results indicate that this hierarchical development results in a total planning effort which is $O(\rho)$ in the number of path segments, ρ . This system therefore serves as a working example which exhibits performance consistent with the theoretical results derived by [Kleinrock and Kamoun, 1977] and [Korf, 1987] which indicate that it is possible to achieve $O(\rho)$ performance by planning hierarchically.

As an agent moves through an environment, however, errors and changes in the environment which cause replanning can become commonplace. Using experimental data presented in Chapter 6 it is argued that replanning can be frequent enough to cause the total planning effort for a coarse grained system to exhibit $O(\rho^2)$ behavior, whereas the fine grained RML system will exhibit a complexity of $O(\rho)$. This supports the central claim of the thesis for the planning component of the system.

Section 5.1 describes sketch-a-plan in detail and shows how it uses scope and perspective to focus the planning computations. Following this discussion, Section 5.2 presents analytical arguments showing significant savings resulting both from the specific sketch-a-plan algorithm and its use in the RML architecture. Section 5.3 offers experimental evidence in support of the complexity arguments of Section 5.2 in "typical" situations and argues that the resulting plans are reasonable.

5.1 Sketch-a-Plan

The objective for sketch-a-plan is to refine a goal under consideration without going into too much detail. The reason for not sketching plans in great detail is to minimize the cost of replanning when the

agent fails to achieve a subgoal. Given the goal of going from Amherst, Massachusetts to Palo Alto, California, for example, sketch-a-plan should not generate a plan sketch of the form:

(and (turn 3.5 degrees) (move 2.4) ...)

Generating a plan like this is computationally expensive and, due to the realities of the agent's effectors and the environment, it would almost certainly be incorrectly executed, resulting in an equally expensive replanning effort when the error is discovered. Plan sketches should be abstract in the sense of being a least commitment refinement. A better plan sketch might be:

(and

(ptrans Amherst-Massachusetts Chicago-Illinois)

(ptrans Chicago-Illinois San Francisco-California)

(ptrans San Francisco-California Palo-Alto-California)).

The implementation of sketch-a-plan used in this study builds on the A* algorithm [Hart, Nilsson and Raphael 1968, 1972], taking advantage of the representation of space described in chapter 4 to provide the means for deciding the level of detail generated in the resulting plan sketch. As will be seen, the representation is also used in a way which serves as a focusing mechanism, providing dramatic improvement in planning costs. Sketch-a-plan is described in 5.1.3. First, however, it will be helpful to describe how A* has been used in its implementation.

5.1.1 A* and planning costs

The A* algorithm (Figure 5.1), which originally appeared in [Hart, Nilsson and Raphael, 1968, 1972], is a graph search algorithm which has often been used for path planning [Raphael, 1976], [Lozano-Perez and Wesley, 1979] and [Arkin, 1987]. Given a graph G and a cost function g defined on adjacent nodes in G , A* is guaranteed to find the least cost path between any two nodes in G . Moreover, A* is optimal in the sense that it finds this path by examining the smallest number of nodes necessary to guarantee that the path is a minimum cost solution [Hart, Nilsson and Raphael, 1968].

A* is efficient by these standards, but its growth characteristics tend to be exponential in the number η of nodes. The reason for this can be traced to step 5 (Figure 5.1). In the absence of focusing mechanisms the successor function will return a number of nodes proportional to the total number η of nodes in the universe. Each of these nodes must be considered in step 6 to determine whether or not it is

in C, G ... etc. The loop defined by steps 2-8 must be repeated at least ρ times, where ρ is the length of the resulting path. Hence the complexity¹ is at least $\Omega(\eta^\rho)$. On the other hand, A* can be no worse than exhaustive search so its complexity is $O(\eta^\rho)$. Hence the complexity of A* is $\Theta(\eta^\rho)$. This exponential growth is characteristic of state space search based planning algorithms [Hendler, Tate and Drummond, 1990].

1. Create:
 - search graph $G := \{S\}$
 - open nodes $O := (S)$ ordered list
 - closed nodes $C := \{\}$
2. If $O = \text{nil}$ then exit with failure
3. Select n as the first element in O and set:
 - $O = \text{remove}(n, O)$
 - $C = \text{union}(C, \{n\})$
4. If n is a goal then return path
5. Let
 - $M = \text{successors}(n)$
 - $G = \text{union}(G, M)$
6. Update
 - For each m in M
 - if m is not in G
 - establish a pointer from m to n
 - add m to O
 - else
 - redirect pointer of m to n , if appropriate
 - if m is in C redirect pointers of its descendents if appropriate.
7. Order O in increasing values of $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$
8. Go to 2.

Figure 5.1 The A* Algorithm summarized from [Nilsson 1980].

Close inspection of Figure 5.1 reveals several factors which strongly influence how efficiently A* can perform on a given problem:

1. The way the problem is interpreted as a graph has a potentially strong influence on the complexity of the resulting algorithm. It is important that the graph be as sparse as possible, without sacrificing plan quality.

¹ When describing the complexity of a function f the following notation is used:

$f(n) = O(g(n))$ means there are constants α, k such that for all $n > k$ $f(n) \leq \alpha * g(n)$,

$f(n) = \Omega(g(n))$ means there are constants α, k such that for all $n > k$ $\alpha * g(n) \leq f(n)$,

$f(n) = \Theta(g(n))$ means there are constants α, k such that for all $n > k$ $\alpha * g(n) \leq f(n) \leq \alpha * g(n)$.

2. The cost function g and the heuristic function \hat{h} ($\hat{f} = \hat{g} + \hat{h}$) must be specified and must be efficiently computable.

3. The successor function should return as few nodes as possible without sacrificing plan quality. In addition to a well chosen representation for the problem (few total nodes in the graph) the successor function should choose nodes in a focused manner.

The next two sections describe how these factors are handled in sketch-a-plan. In this discussion it will be shown how the locale network is used to control the complexity of A* by providing several focusing mechanisms. The resulting complexity will be discussed in 5.2.

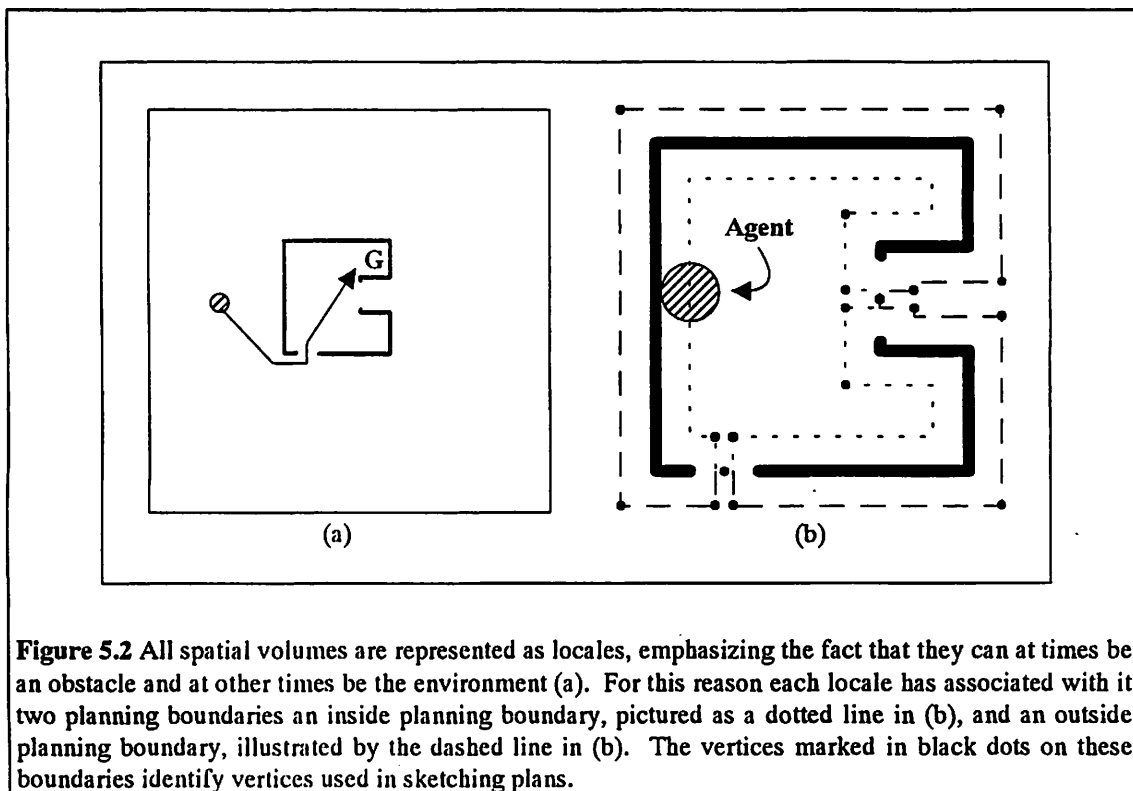


Figure 5.2 All spatial volumes are represented as locales, emphasizing the fact that they can at times be an obstacle and at other times be the environment (a). For this reason each locale has associated with it two planning boundaries an inside planning boundary, pictured as a dotted line in (b), and an outside planning boundary, illustrated by the dashed line in (b). The vertices marked in black dots on these boundaries identify vertices used in sketching plans.

5.1.2 The graph, the heuristic function and some terminology

The state space graph used by sketch-a-plan is motivated by the often used space described by [Lozano-Perez and Wesley, 1979]. Obstacles are all modeled as polygons and these obstacles are grown into the surrounding free space by one half the width of the agent. The vertices of these new polygons become the nodes in the graph and arcs connect vertices if there is a straight free path between them. Representing the problem in this way has the advantage that clear path calculations can be done rather efficiently because the agent can be treated as a single point. This simplifies deciding whether or not a path is free to determining whether or not that path intersects any of these "grown" boundaries.

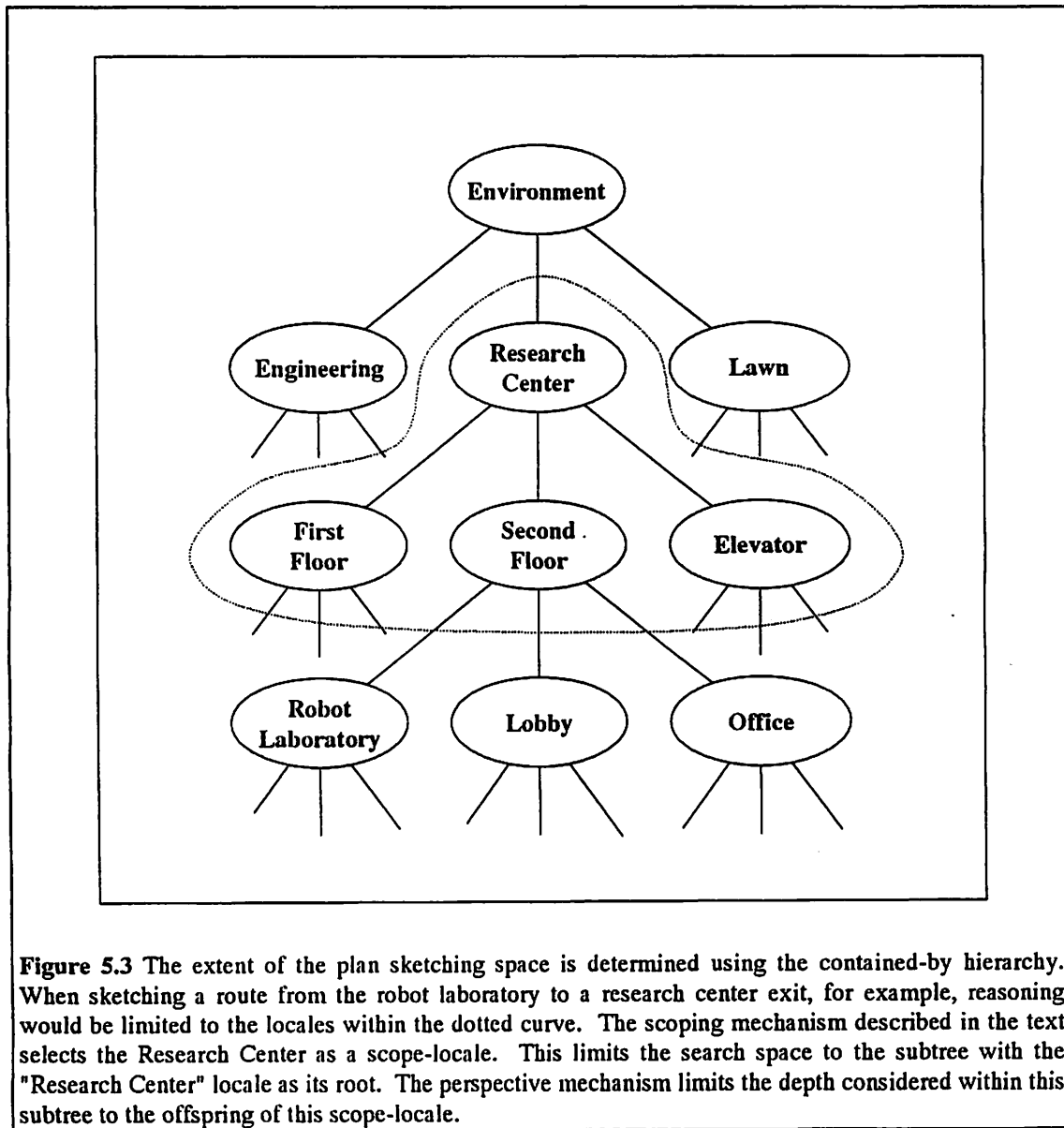
The nature of the locale representation lends itself well to this concept, but the uniform treatment of spatial entities makes some modification necessary. It is possible in this system of representation for the agent to sometimes be inside a spatial entity and at other times be outside of it (Figure 5.2 (a)). For this reason locales are given two such planning boundaries: an inside set and an outside set (Figure 5.2 (b)). It is from these boundaries that planning points, the nodes in the graph, are determined. The inside planning points consist of all the vertices in the inside planning boundary which are associated with a convex corner as viewed from the inside of the locale. The outside planning points consist of all vertices of the outside planning boundary which are associated with corners which are convex when viewed from the outside. In addition to these planning points certain points, designated doorway points, are associated with the locale. Each of these corresponds to the midpoint of a passable door (Figure 5.2 (b)). These planning boundaries and planning point sets are computed once for each locale, are expressed in the coordinate system of that locale, and are permanently stored in the properties slot of the locale frame (Figure 4.1). They are modified only if new information about the properties of the locale itself is entered into the model. If a locale (a desk for example) were moved its planning boundaries move with it, making it unnecessary to recompute them.

The heuristic function used in the experiments was the Euclidean metric implemented to work with the local coordinate systems of the locale hierarchy. A number of heuristic functions could have been used in this system. Since a considerable amount of planning is done inside buildings, a city block measure could be used in those locations to some benefit. The heuristic function could also be modified to reflect the type of terrain found on the traveling surface for various choices of path. These functions have been explored by others [Arkin 1987]. Incorporating such ideas into the system described in this dissertation are straightforward.

5.1.2 The Successor Function

The choice of a successor function is of particular importance because it has the potential of keeping the planning complexity low. The choice of successors to a node in sketch-a-plan is focused by three mechanisms: a scoping mechanism, a perspective mechanism and a semantic filter. The scoping and perspective mechanisms, described in Section 4.1.3, are used to reduce the number of nodes considered

by specifying the portion of the environment and level of detail considered by plan sketching. Semantic filtering further reduces the number of nodes considered using a case-based analysis of what types of nodes may be useful in a particular situation.



The scoping mechanism limits the space under consideration to a small number of locales, using the contained-by hierarchy of the locale network. This mechanism, which is a part of the sketch-a-plan algorithm described in Section 5.1.3, determines the smallest locale in which the desired plan sketch can be constructed. This locale is called the scope-locale. The successor function limits its search for

successors to a subtree of the contained-by hierarchy with the scope-locale as its root (Figure 5.3). This limits the *volume* of the space considered while reasoning about paths. When sketching a plan to get from the robot laboratory to a research center exit, for example, the scope-locale would most likely be the research center locale. Information about San Francisco, Paris and Saturn would not be considered.

The perspective mechanism, which is a component of the successor function, limits how deeply reasoning probes into the subtree defined by the scope-locale. Continuing the example of the previous paragraph, it would be important to reason about the floors, stairways and elevators of the research center; but it would be important to be able to ignore the eraser (a locale) in the top left drawer (the parent locale of the eraser) of Allen Hanson's desk (the parent locale to the drawer). The successor function described in this section limits reasoning about routes to the scope-locale and its immediate offspring (Figure 5.3).

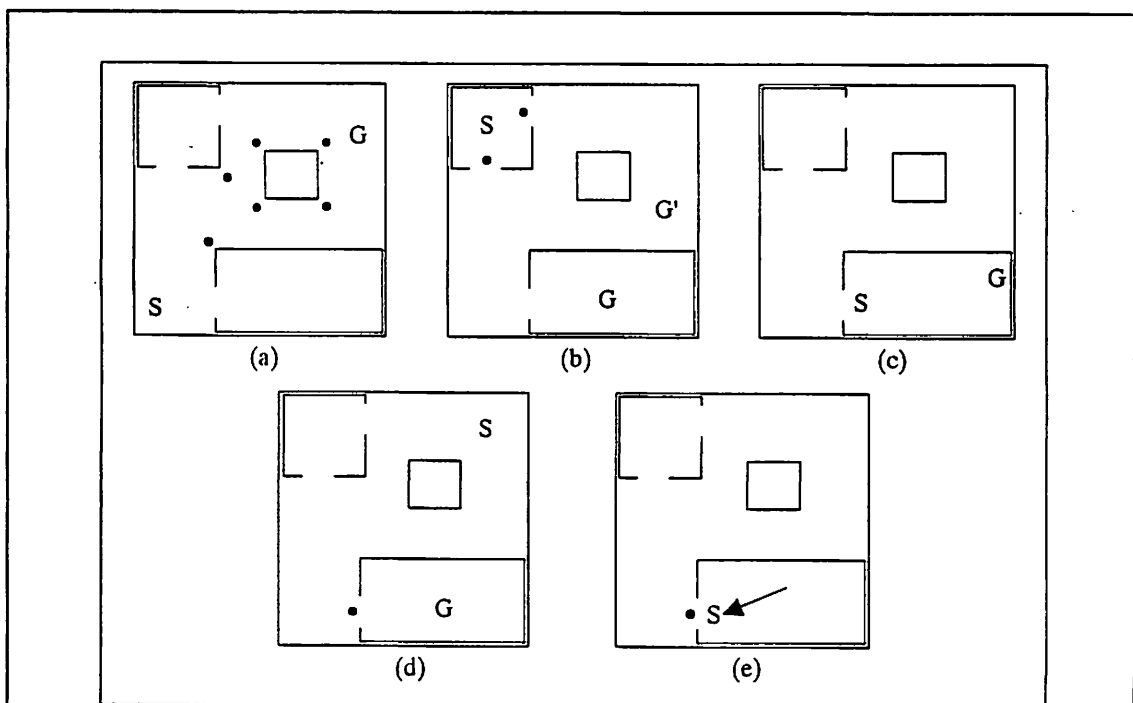


Figure 5.4 Successors to a start location, S, are determined in part by the situation. These situations are illustrated here, using the example of a locale with four sublocales: free-space, two buildings and a fountain. In (a) both S and the goal location, G, are in the *free space* of the locale; in (b) S is in one of the buildings (not in free space) while the goal is in a different sublocale, either at G or G'; (c) represents the situation where S and G are in the same non-free-space sublocales; in (d) S is in free space while G is not; and (e) represents the situation where S is the result of a move with intent to pass through a door.

successors (start-location goal-location scope-locale)

1. Let sub-locales := the offspring locales of scope-locale
start-sublocale := the member of sub-locales which contains start-location
goal-sublocale := the member of sub-locales which contains goal-location
2. If (start-location is of type door-way-entrance) then
return door-way points of start-sublocale
else if (start-location and goal-location are both in the free-space sublocale) then
if (the straight line path from start-location to goal-location is unobstructed) then
return goal-location
else candidates := union {(inside-nodes of scope-locale) (outside-nodes of sub-locales)}
return all candidates to which the straight line path from start-location is unobstructed.
else if (start-sublocale \neq free-space and start-sublocale \neq goal-sublocale) then
return start-sublocale door-way points
else if (goal-sublocale is not free-space and start-sublocale is free-space) then
return goal-sublocale door-way points
else start-sublocale = goal-sublocale and neither is free-space
return goal-location

Figure 5.5 The successor function algorithm.

Semantic filtering is also performed by the successor function. As described in section 5.1.2 planning points have been divided into three categories: inside-planning-points, outside-planning-points and door-way-planning points. This distinction has been made because the semantics of the goal under consideration offers a simple, computationally inexpensive means for further reducing the number of nodes returned by the successor function. The selection of nodes is based on the relationship between the starting location S and the finish location G according to the following case analysis:

1. Both S and G are in the free space F of the scope-locale (Figure 5.4a). In this case the path to be generated will stay within F and it is only necessary to avoid obstacles and the walls of the scope-locale. G will suffice as the only successor, if the path from S to G is unobstructed. Otherwise the set of successors consists of the reachable inside-planning-points of the scope-locale and the reachable outside-planning-points of its offspring.

2. S is in an offspring locale, L1, of the scope-locale (not the free-space sublocale) and G is in a different offspring locale, L2, (possibly the free-space sublocale) of the scope-locale (Figure 5.4b). In this case the objective is to get out of L1. The successor nodes will consist of the door-way points of L1. Since reasoning is limited in detail by the perspective mechanism, obstacles within L1 are ignored for the time being. The recursive construction of plan-and-monitor will force a refinement at a later time.

3. S and G are in the same offspring locale, L, (not free-space sublocale) of the scope-locale (Figure 5.4(c)). In this case analysis is deferred as too detailed for the current plan sketch. G is returned as the only successor. As was the situation for case 2, the path from S to G is considered reachable for the time being.

4. S is in the free-space offspring-locale, F, of the scope-locale and the G is in a different offspring-locale, L, of the scope-locale (Figure 5.4(d)). In this case successors will consist only of the doorway-points of the sublocale L.

5. S is a door-way-exit (Figure 5.4(e)). A location is only marked as a door-way-exit when it has been previously generated by this function as a means for leaving a sublocale. The only sensible successor then is the "matching" doorway-point in the connecting sublocale. This is by definition reachable whenever the door is open.

The successor function is summarized in Figure 5.5. By following the reasoning in steps 1 and 2 of the figure it is clear that all reasoning is done in terms of the scope-locale and its immediate offspring. This implements the perspective mechanism. Semantic filtering is implemented by the conditional statements in step 2.

```
sketch-a-plan (goal)
  1. Let scope-locale := smallest locale containing the goal start and goal finish
  2. Let plan := A*(goal, scope-locale)
  3. If plan = nil
      then let scope-locale := parent(scope-locale)
           restructure the scope-locale.
           if scope-locale = 'environment
               then return "failure"
           else go to 2.
      else return plan.
```

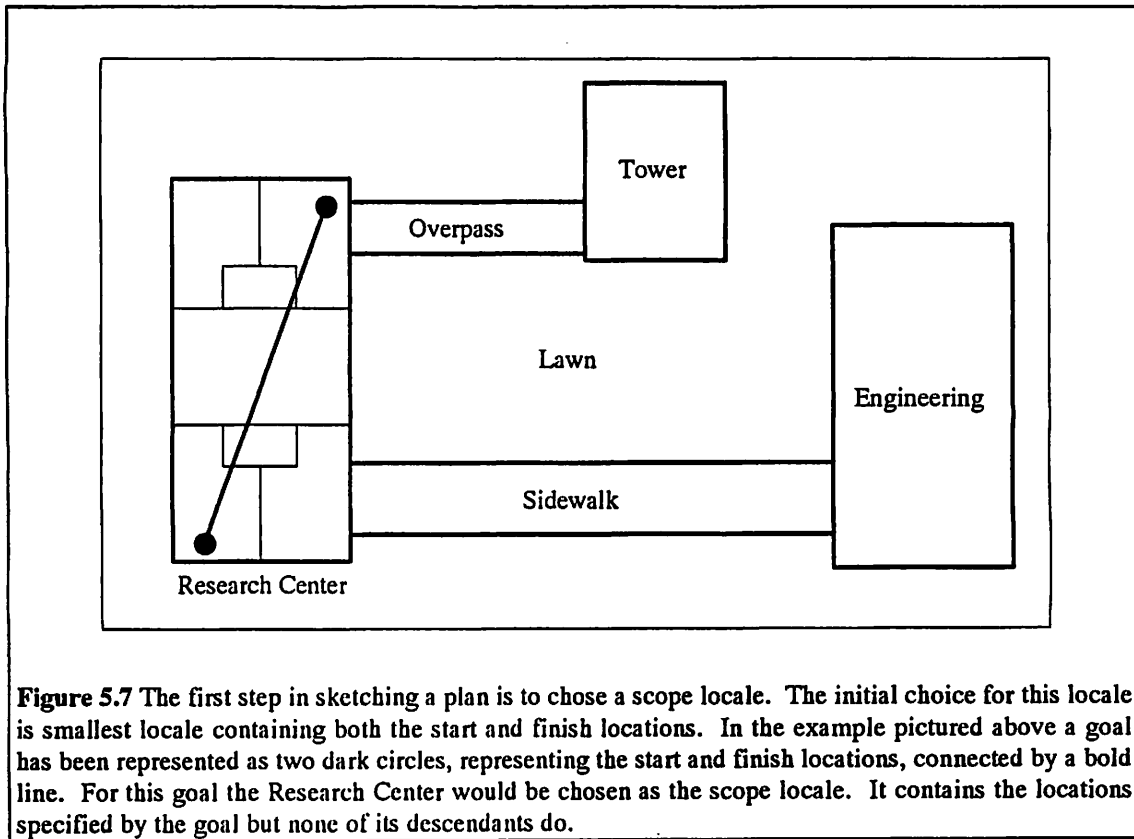
Figure 5.6 The sketch-a-plan algorithm.

5.1.3 The Sketch-a-Plan Algorithm

Armed with the machinery described in the previous sections, the algorithm for sketch-a-plan is relatively straightforward to describe (Figure 5.6). Given a goal:

(ptrans start-location finish-location)

sketch-a-plan first chooses the scope-locale. The first approximation to this locale is the smallest locale in the contained-by hierarchy which contains both the start-location and the finish-location of the goal (Figure 5.7). A* is then applied to the nodes generated by the successor function which, by construction, will not include any nodes outside of the scope-locale. If A* is successful in finding a path, it is returned as the plan sketch (as sequence of goals). It is possible, however, that A* will be unsuccessful if this scope-locale is too small to contain a free path (Figure 5.8). In this case A* must be reinvoked using a larger scope-locale.



When a locale network properly describes the environment this situation will not arise. Locales should be constructed so that the free space within them is connected. This can always be done by splitting a locale into sublocales. The problem introduced by the barrier in Figure 5.8, for example, can be solved by restructuring the network. The locale, C, in Figure 5.8 (a) can be split into two locales, C_1 and C_2 as shown in Figure 5.8 (b). C is then replaced as an offspring locale to its parent by C_1 and C_2 . Offspring locales of C to the left of the barrier will become offspring of C_1 ; the others will become offspring of C_2 . In Figure 5.6 this process is referred to as restructuring the scope-locale.

Once the scope-locale has been restructured, its parent becomes the new scope-locale and planning with A* continues. Each time A* fails to find a solution sketch-a-plan continues to working its way up the contained-by hierarchy until it either finds a solution or exhausts the possibilities. In the latter case it stops and indicates that it has failed (no solution exists, given what is known about the environment).

Since, as stated in Chapter 1, it was assumed for this dissertation that the environment was static and that all obstacles were known, the detailed scope-locale restructuring algorithm was not actually written.

The environment was properly modeled from the beginning. The extensions of plan-and-monitor and sketch-a-plan necessary to cope with dynamic and unknown environments has been left as a matter for future research.

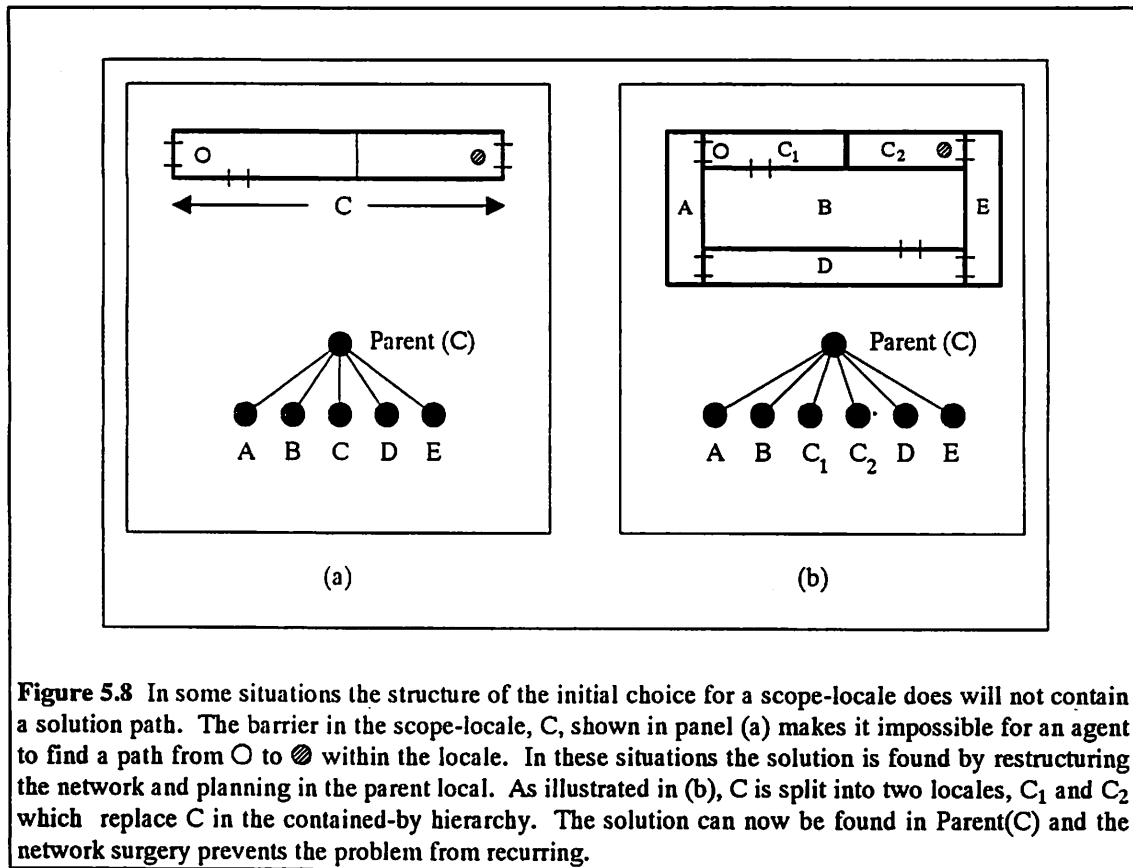


Figure 5.8 In some situations the structure of the initial choice for a scope-locale does will not contain a solution path. The barrier in the scope-locale, C, shown in panel (a) makes it impossible for an agent to find a path from O to ⊙ within the locale. In these situations the solution is found by restructuring the network and planning in the parent local. As illustrated in (b), C is split into two locales, C₁ and C₂ which replace C in the contained-by hierarchy. The solution can now be found in Parent(C) and the network surgery prevents the problem from recurring.

5.2 Remarks about planning complexity

It has been claimed that the RML strategy and the focusing methods described in this chapter result in substantial savings in planning computations and yet generate reasonable plans. This section substantiates the claim concerning computational savings for the *worst case* situation using analytical arguments. Section 5.3 supports the complexity claim for "typical" situations and shows that the resulting plans are "reasonable" by presenting experimental results.

To simplify the discussion, the analysis of planning complexity has been divided into three sections. The first of these, Section 5.2.1, begins by deriving an expression for the complexity of a single invocation of sketch-a-plan. The result is used in Section 5.2.2 to determine the complexity of planning

an entire route using sketch-a-plan and plan-and-monitor. These two sections show that the effect of the scope and perspective mechanisms is to induce a hierarchical plan development, resulting in a total planning effort of $O(\rho)$, under the assumption that the environment is static and there are no errors in execution. Section 5.2.3 analyzes planning in conjunction with action in dynamic environments. Using the complexity results from plan-and-monitor, it is argued that the RML paradigm reduces planning complexity from $O(\rho^2)$, when planning acts as a macroprocess, to between $O(\rho \log(\rho))$ and $O(\rho)$, when planning acts as a part of a fine grained RML loop.

5.2.1 Complexity of plan sketching

One of the effects of representing an environment in terms of locales is to set up a hierarchy of space partitions. This partitioning is a feature of the "locale hierarchy" described in chapter 4. The environment locale is partitioned into sublocales and each of these sublocales is itself partitioned into sublocales. This partitioning process is not uniquely defined. There are a number of possible partitions for each environment. The earth, for example, might be partitioned into hemispheres: the eastern and western hemispheres; or it might be partitioned into the continents and the oceans. The partitioning is a matter of choice.

It is always possible to represent an environment of η nodes as a locale network in such a way that, for given κ and β , the number of nodes associated with each locale in the hierarchy is approximately κ and the branching factor in the locale hierarchy is β . When represented this way the total number η of nodes in the environment is:

$$\eta = \kappa*(1+\beta +\beta^2 + \beta^3 + \dots + \beta^{\lambda-1}) = \kappa*(\beta^\lambda - 1)$$

where

$$\lambda = \log_{\beta} \left(\frac{\eta}{\kappa} \right)$$

The scope and perspective mechanisms described in 5.1.2 dictate that the sketch-a-plan algorithm only consider nodes associated with the scope-locale and the offspring locales of that scope-locale (as in the example shown in Figure 5.3). As a result, the portion of the environment considered by an invocation of

sketch-a-plan is limited to $\kappa*(1+\beta)$ nodes, rather than the entire space η of nodes. The fraction of the space considered is therefore:

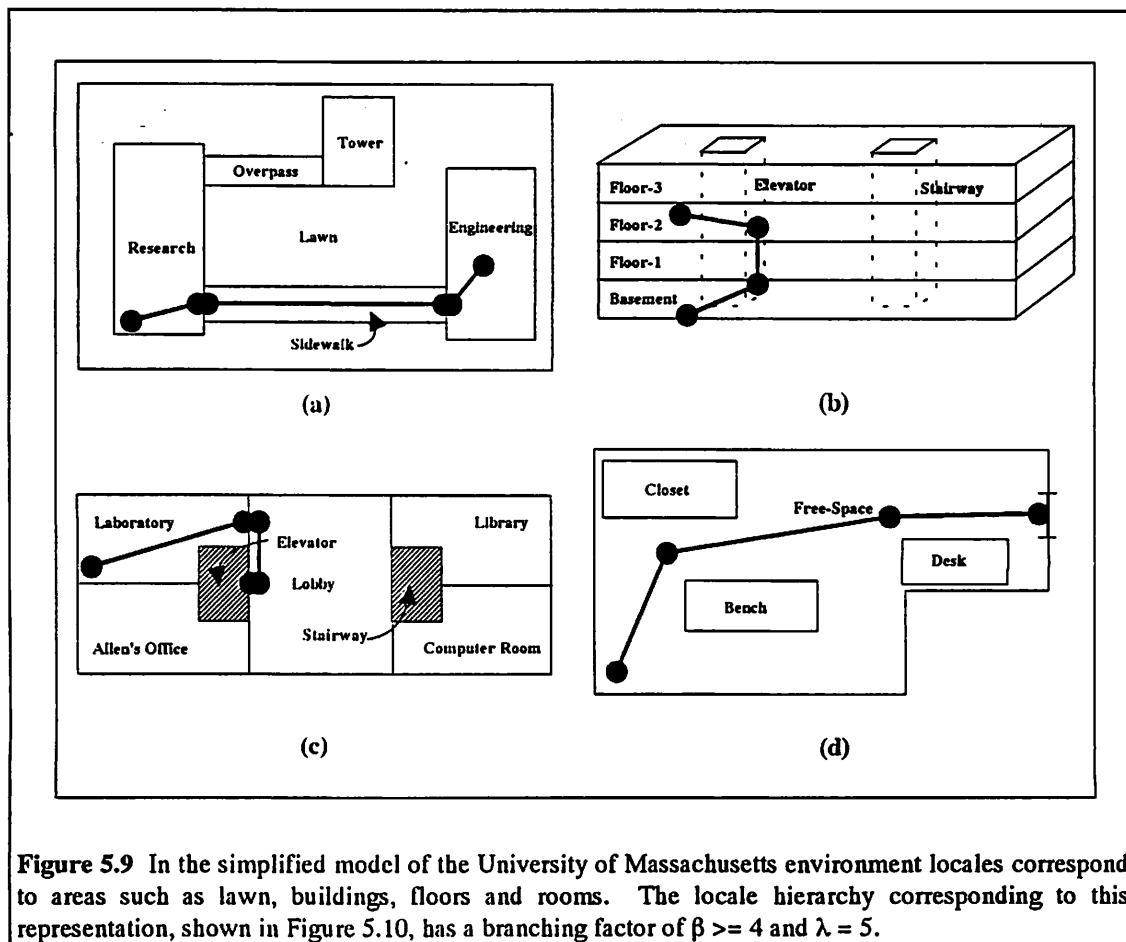
$$\frac{\kappa*(1+\beta)}{\kappa*(\beta^\lambda - 1)} = \frac{1+\beta}{\beta^\lambda - 1}$$

Due to the η^ρ worst case behavior of A* this has a strong effect on the complexity of a single invocation of sketch-a-plan. The ratio of the complexity for sketch-a-plan with focusing $(\kappa*(1+\beta))^\rho$ to that without the focusing mechanisms $(\kappa*(\beta^\lambda))^\rho$ is on the order of:

$$\frac{(\kappa*(1+\beta))^\rho}{(\kappa*(\beta^\lambda))^\rho} = \frac{\kappa^\rho * (1+\beta)^\rho}{\kappa^\rho * (\beta^\lambda)^\rho} = \frac{(1+\beta)^\rho}{\beta^{\rho\lambda}}$$

As β gets large this expression becomes:

$$\frac{(1+\beta)^\rho}{\beta^{\rho\lambda}} \text{ -----} \rightarrow \frac{\beta^\rho}{\beta^{\rho\lambda}} = \frac{1}{\beta^{\rho(\lambda-1)}}$$



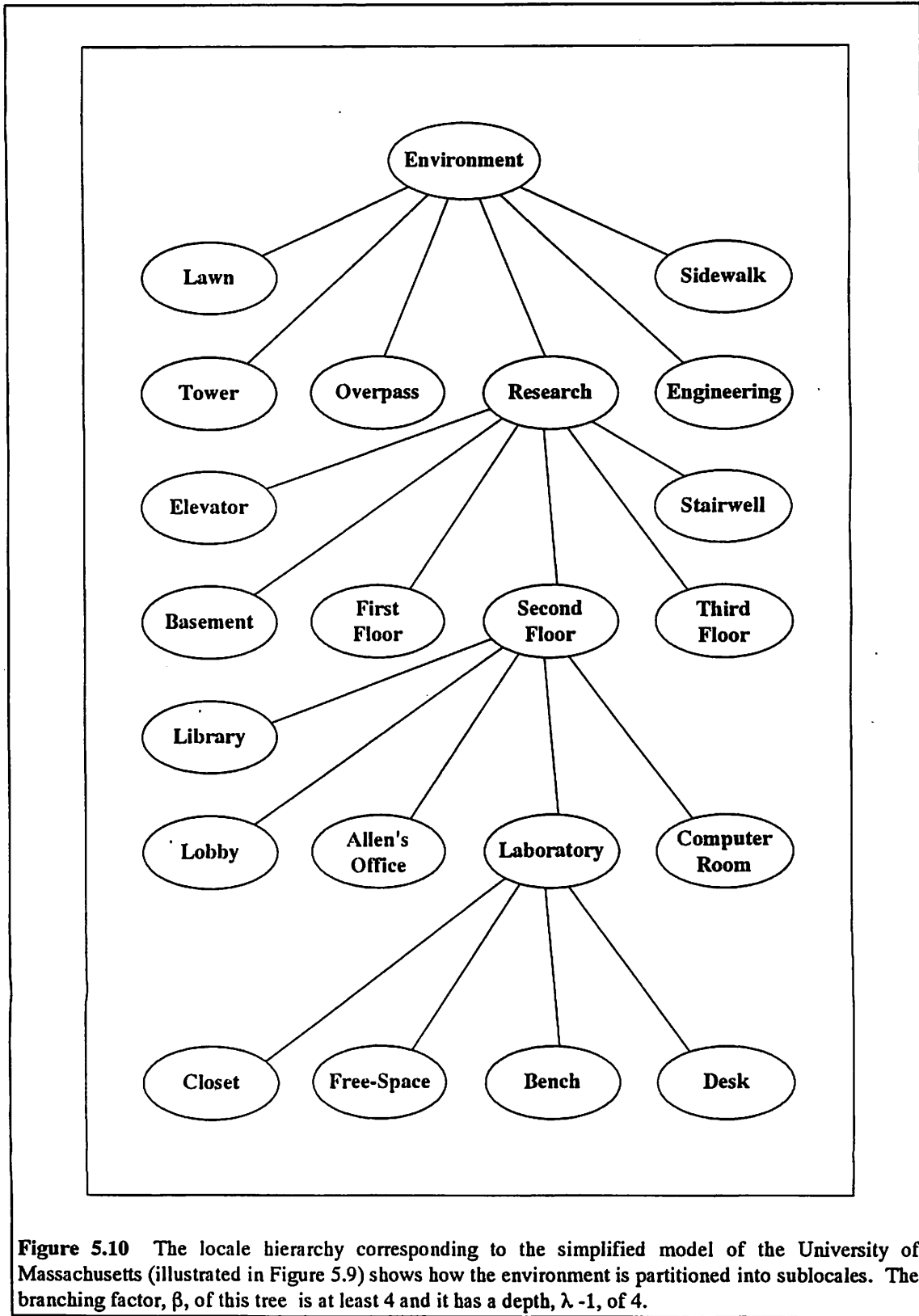


Figure 5.10 The locale hierarchy corresponding to the simplified model of the University of Massachusetts (illustrated in Figure 5.9) shows how the environment is partitioned into sublocales. The branching factor, β , of this tree is at least 4 and it has a depth, $\lambda - 1$, of 4.

As this complexity ratio indicates, the reduction in plan sketching complexity due to the use of the scope and perspective mechanisms can be very significant. An example will illustrate the magnitude of this savings. Consider the simplified UMass environment shown Figure 5.9. In this case $\lambda = 5$ and $\beta \geq 4$ so the ratio of the number of nodes considered in by sketch-a-plan with focusing to that without is:

$$\frac{1+4}{4^5-1} = 5 \times 10^{-3}$$

For a typical path length of 20 segments this means that the complexity ratio for sketch-a-plan (focused/unfocused) in this example is:

$$\frac{(1+4)^{20}}{4^{20 \times 5}} = \frac{5^{20}}{4^{100}} = 5.9 \times 10^{-47} = \frac{1}{1.7 \times 10^{46}}$$

The unfocused sketch-a-plan would do 1.7×10^{46} times as much computation the focused one. This argues well for a significant potential savings gained by using the focusing mechanisms described in this chapter when using an algorithm like A* for planning. In the RML scheme, however, sketch-a-plan is invoked many times to create a plan.

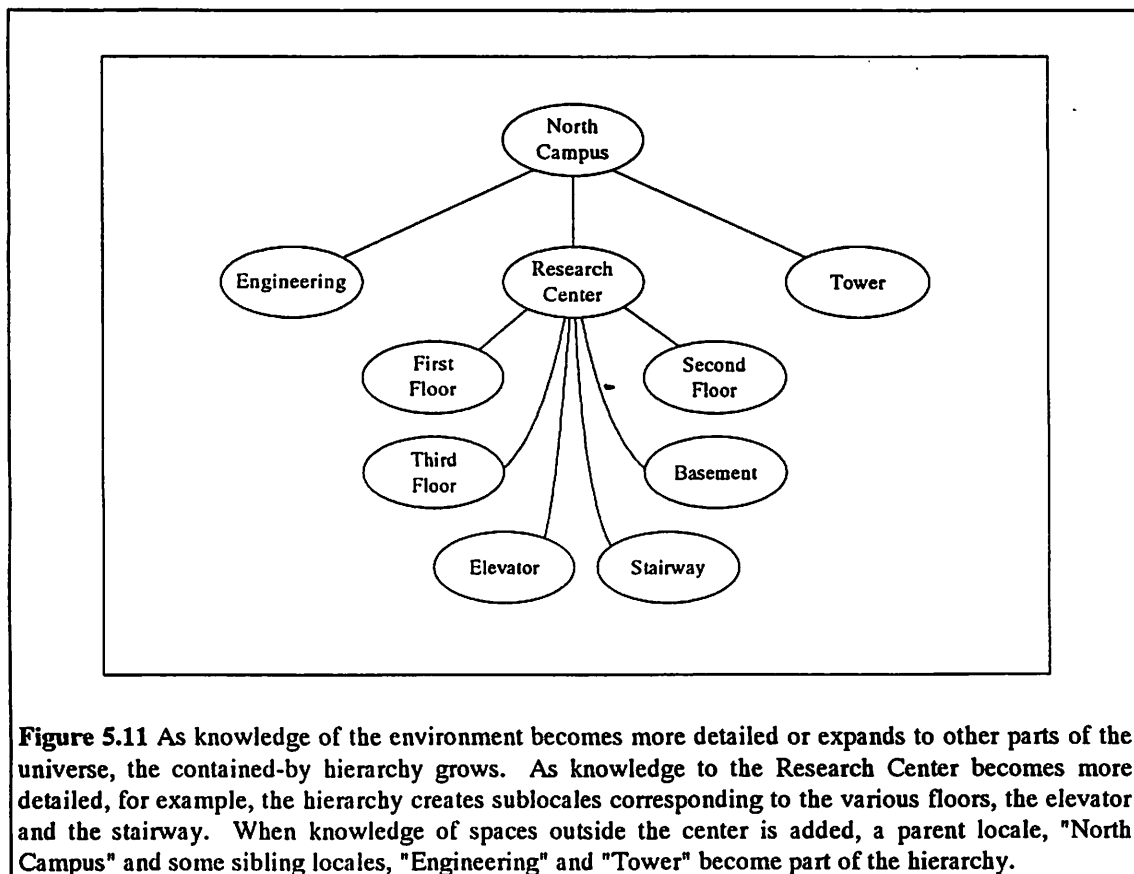


Figure 5.11 As knowledge of the environment becomes more detailed or expands to other parts of the universe, the contained-by hierarchy grows. As knowledge to the Research Center becomes more detailed, for example, the hierarchy creates sublocales corresponding to the various floors, the elevator and the stairway. When knowledge of spaces outside the center is added, a parent locale, "North Campus" and some sibling locales, "Engineering" and "Tower" become part of the hierarchy.

5.2.2 Total planning complexity assuming a static environment and no execution errors

Assuming, for the moment, that the environment is static and that each primitive subgoal is exactly achieved, it is possible to determine the complexity of the total planning effort using the plan-and-monitor and sketch-a-plan when no replanning is necessary. Under these assumptions the total planning effort will be the cost of producing one detailed plan. To determine the complexity of this task let:

b = the number of subgoals returned by a single invocation of sketch-a-plan for an input goal, and

ρ = the number of subgoals in the final path.

In terms of these quantities the total number of times that sketch-a-plan is called for the plan is

$$1 + b + b^2 + \dots + b^{\log_b(\rho) - 1} = \frac{b^{\log_b(\rho)} - 1}{b - 1}$$

so the complexity of the total planning effort is

$$\Pi = (b^{\log_b(\rho)} - 1) * (\kappa * (1 + \beta))^b = (\rho - 1) * (\kappa * (1 + \beta))^b$$

Since κ , β and b are fixed, this indicates that the total planning effort for plan-and-monitor is linear with respect to path length, ρ , rather than exponential. Total planning complexity is $O(\rho)$. This is consistent with the results derived by [Kleinrock and Kalmoun, 1977] and [Korf, 1987] which show that it is theoretically possible to achieve linear planning complexity if planning is done hierarchically. Although the plans generated in the RML scheme are not developed in a strict hierarchical order, the complexity of producing the resulting plan under the assumptions of this section is the same as if it were.

It is worth noting that the complexity of the planning effort can be kept under control as the size of the environment increases. Planning complexity is a function of the *local* complexity (node density κ) of the environment, not the *global* complexity of the environment (η). In the expression:

$$\Pi = (\rho - 1) * (\kappa * (1 + \beta))^b$$

$\kappa * (1 + \beta)$ can be kept bounded by partitioning the environment appropriately. As the total number of nodes (η) increases, the locale hierarchy will be organized accordingly (Figure 5.11).

To illustrate the magnitude of the computational savings we can compare A* with plan-and-monitor on our example environment. The ratio of the complexities is:

$$\frac{(\rho - 1) * (\kappa * (1 + \beta))^b}{(\kappa * (\beta^\lambda - 1))^P} = \frac{(\rho - 1) * \kappa^b (1 + \beta)^b}{\kappa^P (\beta^\lambda - 1)^P}$$

In the environments with which we have experience, sketch-a-plan typically returns 5 subgoals so $b=5$, $\kappa=10$, $\beta=4$, $\lambda=5$. For a path length of 20 this means the ratio of the total complexity for plan-and-monitor to that for A* would be:

$$\frac{(20 - 1) * 10^5 * (1+4)^5}{10^{20} * (45-1)^{20}} = \frac{19 * 3.1 * 10^8}{1.6 * 10^{80}} = 3.7 * 10^{-71}.$$

For this example, A* would require on the order of $2.69 * 10^{70}$ times as much computation as plan-and-monitor.

This argument clearly indicates that plan-and-monitor working with sketch-a-plan offers dramatic savings in computation over an algorithm like A*. The result, however, is expressed as a function of the number of segments in the final path. To be complete it is necessary to examine the relationship between the number of path segments ρ_A , generated by A* for a goal, and the number of path segments ρ_M , generated by plan-and-monitor for the same goal. Clearly if $\rho_M = \rho_A * \eta^p$, nothing has been gained. The experience with paths generated during experimentation (Section 5.3) indicates that this relationship is well behaved. In typical situations the results indicate that $\rho_M \approx \rho_A$. By examining what happens to ρ_M as a locale is divided into sublocales an analytical expression for this relationship can be derived.

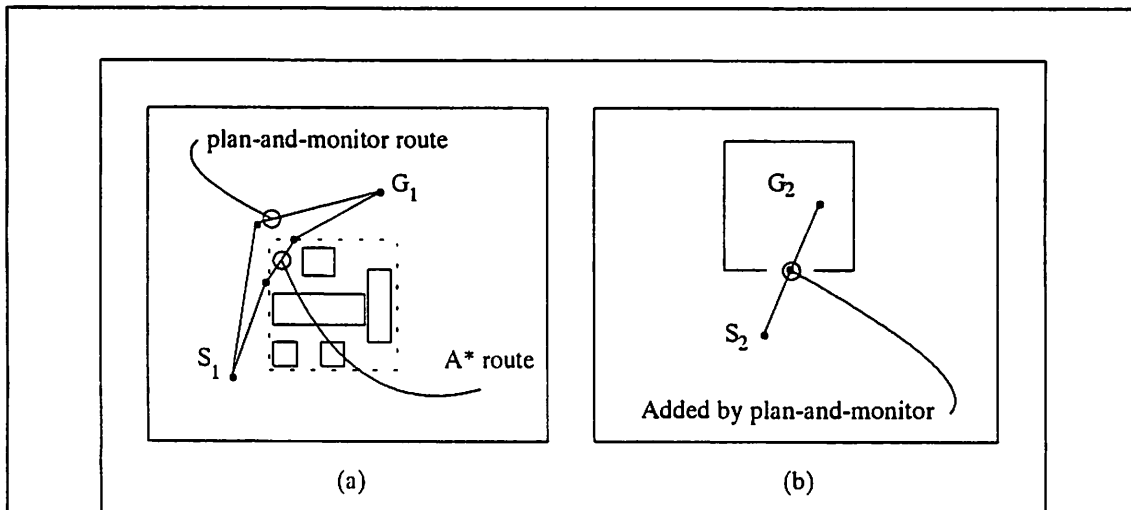


Figure 5.12 The number of path segments, ρ_A , produced by A* for a given goal sometimes differs from the number of path segments, ρ_M , produced by plan-and-monitor for that same goal. When the locale structure is designed to summarize a cluster of obstacles, as is indicated by the dotted in (a), ρ_M will tend to be smaller than ρ_A . When planning a route from one sublocale to another, as illustrated in (b), the plan-and-monitor can add an apparently unnecessary doorway point, causing ρ_M to be larger than ρ_A by one segment.

As illustrated in Figure 5.12, the effect on ρ_M of dividing a locale into sublocales depends on the specific way those sublocales divide space. The number of path segments resulting from plan-and-monitor can be larger or smaller than for A^* . In the interest of placing a bound on the resulting path length, a set of strategies can be used when dividing a locale into sublocales:

1. If a locale L is such that A^* always returns primitive paths of length $b > \rho_A \geq 1$, then it should not be subdivided. A locale with few obstacles, such as the one illustrated in Figure 5.13 (a) would satisfy this condition. In this case $\rho_M = \rho_A$, since the path generated for the locale will be produced by A^* with no recursive calls.

2. If there are a large number of obstacles and they form clusters or if some are contained by others, create sublocales of the locale which correspond to the containers or describe an outline of the clusters. Whenever a container sublocale encloses another sublocale the latter becomes a descendent of the former. This is the situation of Figure 5.13 (b). Desks, bookcases and file cabinets contain obstacles. They become sublocales. The table and chairs, the row of file cabinets and the bookcases can be clustered to simplify the locale. As illustrated in Figure 5.12, this may have the result of shortening the route in the sense that $\rho_M \leq \rho_A$.

3. If there are a large number of obstacles and they do not form clusters, divide the locale into subsets as shown by the dotted curve in Figure 5.13 (c). A forest, for example, can be divided into areas. As illustrated by Figure 5.12, each transition from sublocale to sublocale may create an unnecessary subgoal. As a result, for this type of area $\rho_M \leq \rho_A + \beta$ (where β is the branching factor in the contained-by hierarchy and, consequently the number of sublocales of a locale).

4. If a locale is carved up by walls, as in Figure 5.13 (d), divide it into sublocales defined by these walls. Divide a space only if the doorway between the resulting sections is narrow. In this case the new path segments caused by creating the sublocales will also probably be generated by A^* so $\rho_M = \rho_A$.

5. A long locale, such as the road in Figure 5.13 (e), can be divided into sublocales along its length at places where A^* would usually be required to terminate a path segment. For this case $\rho_M = \rho_A$.

When the cause of the complexity is not due to clusters of obstacles we must divide the locale into sections. In buildings these sections correspond to rooms, locales with solid walls and (usually) relatively narrow doors. As plan-and-monitor has been described, the effect of dividing a locale into rooms is to divide routes into subsequences defined by the doorways between these room sublocales. This has the effect, in some situations, of adding an additional step at each doorway as shown at the right of Figure 5.12. A^* would generate a single step path from S_2 to G_2 , since the path is unobstructed, but plan-and-monitor would generate the intermediate doorway-point, resulting in a two step path. This tends to make $\rho_A \leq \rho_M$. This situation is unusual but, even when it happens, it is unlikely that an agent will be able to

negotiate such a path without error. The chances are high that the doorway point will be part of a replanning effort when A^* is used. Here too, in a practical sense, $O(\rho_M) = O(\rho_A)$.

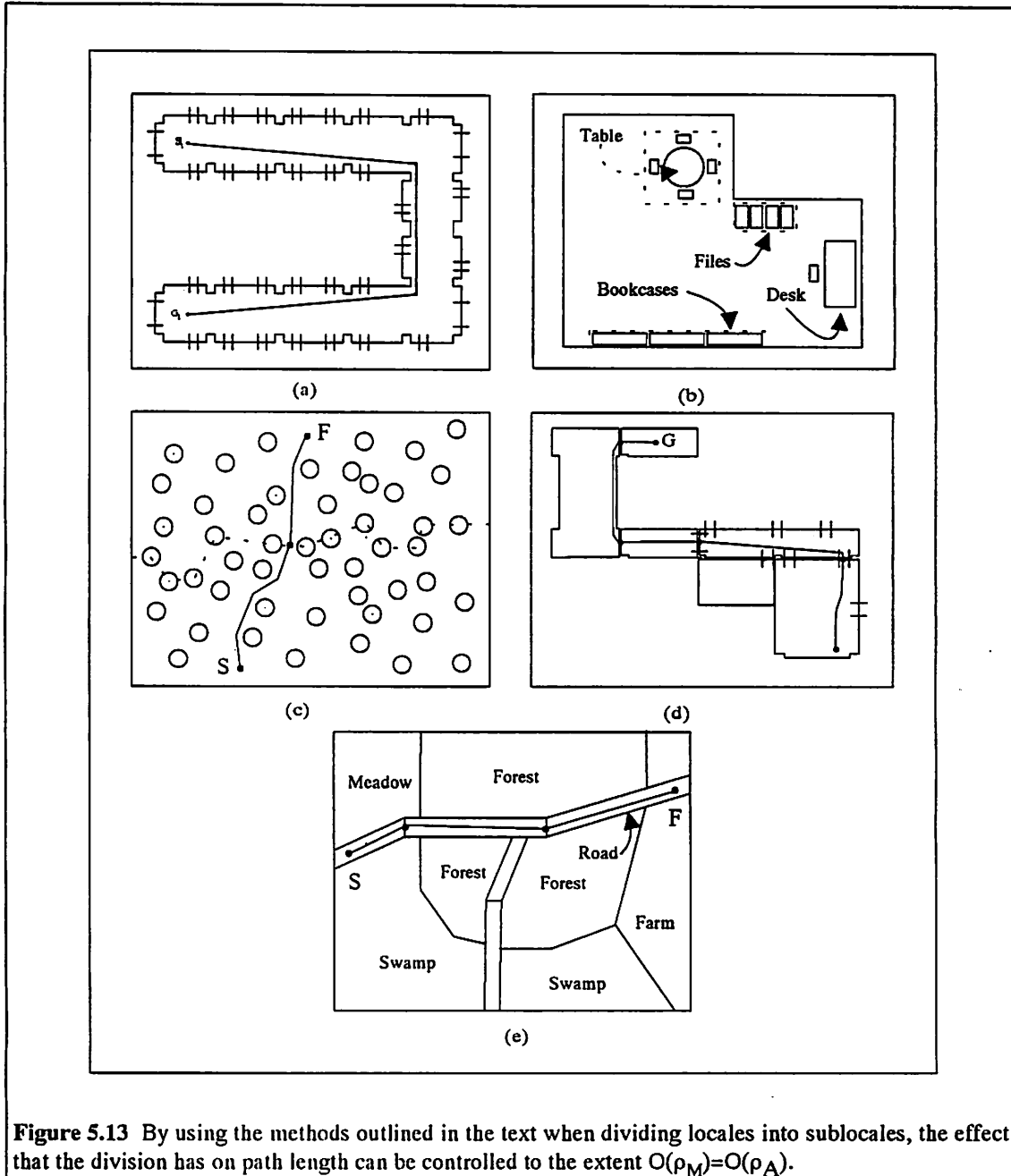


Figure 5.13 By using the methods outlined in the text when dividing locales into sublocales, the effect that the division has on path length can be controlled to the extent $O(\rho_M) = O(\rho_A)$.

The degree to which the length of a plan-and-monitor path will differ from A^* depends on how many levels of the contained-by hierarchy must be descended to reach a sequence of primitive goals. If, for example, a goal g specifies a start S and a goal location G in the locale illustrated in Figure 5.11(a), then

plan-and-monitor will generate its plan using a single invocation to A*. In this case there are no recursive calls to plan-and-monitor, so

$$\rho_{M0}(g) = \rho_A(g)$$

If plan-and-monitor must refine the plan, it may result in a path which has a different length from what would be generated by A*. In cases 1,2,4 and 5 there would be no change. In case 3, however, it is possible that the hierarchical development will add one step per boundary, resulting in a possibility of β additional steps, once for each sublocale boundary crossing. In this case the first call to sketch-a-plan will generate up to β subgoals and the finished plan will refine these subgoals using A*.

$$\rho_{M1}(g) = \rho_A(g_1) + \dots + \rho_A(g_\beta) \leq \rho_A(g) + \beta$$

If each subgoal g_k had to be further refined the situation would repeat: For each g_k

$$\rho_{M1}(g_k) = \rho_A(g_{k1}) + \dots + \rho_A(g_{k\beta}) \leq \rho_A(g_k) + \beta$$

The total resulting plan for two levels of refinement would then be

$$\rho_{M2}(g) = \rho_A(g_{11}) + \dots + \rho_A(g_{\beta\beta}) \leq \rho_A(g) + (\beta + \beta^2)$$

By similar reasoning, if the plan required n levels of refinement the result would be:

$$\rho_{Mn}(g) \leq \rho_A(g) + \beta + \beta^2 + \dots + \beta^n$$

Now, as Figure 5.11(c) indicates, this refinement process terminates when A* would have produced a result. Hence the number of refinements required to construct the plan is $\log_b(\rho_A(g))$. The length of the complete plan for g can be expressed as:

$$\begin{aligned} \rho_M(g) &\leq \rho_A(g) + \{\beta + \beta^2 + \dots + b^{\log_b(\rho_A(g)) - 1}\} \leq \rho_A(g) + \{b^{\log_b(\rho_A(g))} - 2\} \\ &\leq \rho_A(g) + \rho_A(g) - 1 = 2 * \rho_A(g) - 1 \end{aligned}$$

Hence $\rho_M(g) = O(\rho_A(g))$. The computational savings due to plan-and-monitor are in fact dramatic.

5.2.3 The effect of execution errors and dynamic environments on planning complexity

In the previous section it was assumed that the environment was static and that all actions were carried out as planned. When these assumptions are removed, changes in the environment and errors in action execution can make it impossible to continue with a plan. Replanning becomes an issue. Under these conditions the total planning complexity must take into account how often replanning is required and what must be done to reconstruct a plan.

It is difficult to make believable *quantitative* statements about how frequently replanning will be necessary. How accurately an agent executes each action depends on both the capabilities of the agent and on the nature of its environment. Most mobile robots would, for example, produce more errors on a slippery surface than on a non-skid surface. In addition, what constitutes a significant error is problem dependent. An error of one foot may be unimportant when crossing an empty parking lot, but it could be disastrous when moving along a narrow ledge. The frequency of change in an environment is likewise difficult to quantify. Replanning is more likely to be necessary on a city street than on a deserted highway, but it is difficult to say how often in either case. However, based on experiments described in Chapter 6, it is possible to make two *qualitative* statements:

1. Execution errors are likely to be significant when primitive actions are executed without perceptual feedback. It was common for the error resulting from a single primitive action to reach a foot or more. In many environments this would force replanning after every plan step.
2. When primitive actions were executed with perceptual feedback errors were very modest. Errors of less than .5 inch were common. Under these circumstances replanning would not often be required.

Using these statements it is possible to get an appreciation for the total planning complexity.

The effort required to construct each new plan depends on which planning method is used. Following the discussion of Section 5.2.3 the basic ideas embodied in the plan-and-monitor and sketch-a-plan algorithms can be reorganized to build a hierarchical macroprocess planner which would construct a detailed plan in $O(\rho)$ operations. Comparing this macroprocess planning algorithm to the RML planning algorithm in the presence of replanning will show the advantage of the RML architecture when there are execution errors and environmental changes.

The macroprocess architecture, as discussed in Chapter 2, begins by planning a detailed route. When the plan is complete it is executed step by step until something goes wrong or until the uncertainty becomes too high. Then execution stops and perception is used to determine the exact location of the agent and, if necessary, a new plan is generated. Given that it is necessary to replan n times the total planning effort for the macroprocess architecture can be expressed as:

$$\Pi_{MAC} = \{(\rho_0 - 1) + (\rho_1 - 1) + \dots + (\rho_n - 1)\} * (\kappa * (1 + \beta))^b$$

where ρ_k is the number of path segments in the plan resulting from the k th replanning effort.

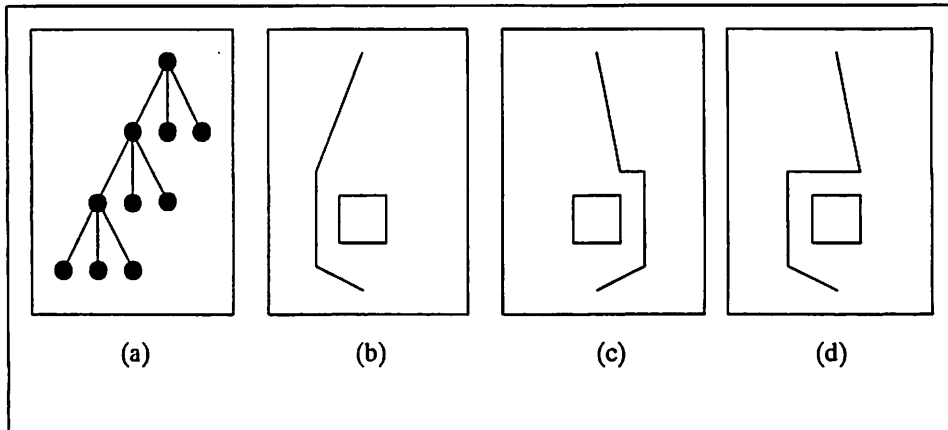


Figure 5.14 The RML architecture develops lazy plans, deferring most of the computation until it is required. Usually the first few subgoals of each plan sketch are primitive (a). Because of this, replanning is likely to be required after each detected error. Otherwise executing the first step of the ideal plan shown in (b) may not result in the smooth recovery of (c) but, rather the more erratic path shown in (d).

The RML architecture develops a plan sketch in a depth first fashion until the first subgoal is primitive. The action begins. Thus action takes place before the plan is fully developed. This has the effect of implementing a lazy planning strategy (meaning that much of the plan is undeveloped) so, when an error or surprise is discovered, the planning process need only continue where it left off to plan the next step. Ideally, this would keep the RML architecture planning complexity to the ideal $O(\rho)$, even in the face of errors and surprises. The plan-and-monitor algorithm does not, however, quite achieve this ideal. If a significant error or surprise occurs some plan sketch reconstruction will be necessary.

Usually the first few subgoals in a plan sketch are developed to the level of primitive subgoals (Figure 5.14 (a)). Because of this, if no plan reconstruction is performed after an error is discovered, the resulting behavior may become erratic. An error in the execution of the original plan of Figure 5.14 (b), for example, may not result in the smooth recovery of Figure 5.14 (c), but rather the plan would attempt to continue by patching the old plan as in Figure 5.14 (d). The only way to be sure of avoiding this is to generate a new plan sketch. The complexity of this process is:

$$\log_b(\rho) * (\kappa * (1 + \beta))^b$$

Hence, if it is necessary to replan m times and if π_k is the number of path segments of a *complete plan* resulting from the k th replanning effort, the total planning effort for the RML architecture can be expressed as:

$$\Pi_{\text{RML}} = \{\log_b(\pi_1) + \log_b(\pi_2) + \dots + \log_b(\pi_m)\} * (\kappa*(1+\beta))^b$$

By construction, if an RML plan were expanded to full detail it would be identical to the plan generated by the macroprocess architecture. This means that $m = n$ and $\pi_k = \rho_k$ for each k . Consequently, $\log_b(\pi_k) < (\rho_k - 1)$ for $1 < \pi_k (= \rho_k)$. So $\Pi_{\text{RML}} < \Pi_{\text{MAC}}$. The RML architecture is more efficient in terms of its total planning complexity. The magnitude of the savings can be seen by observing that:

$$\Pi_{\text{MAC}} = \{(\rho_0 - 1) + (\rho_1 - 1) + \dots + (\rho_n - 1)\} * (\kappa*(1+\beta))^b = O(n * \max(\rho_k))$$

and

$$\Pi_{\text{RML}} = \{\log_b(\rho_1) + \log_b(\rho_2) + \dots + \log_b(\rho_n)\} * (\kappa*(1+\beta))^b = O(n * \log_b(\max(\rho_k))).$$

This savings is due to the lazy planning aspect of the RML architecture.

By making a two assumptions it is possible to obtain a more intuitive appreciation for the total complexity of RML and macroprocess planning. The first assumption is that replanning occurs after the execution of each plan segment. As mentioned earlier in this section, this is not an unreasonable assumption in the light of the experiments discussed in Chapter 6. The second assumption is that the number of plan steps ρ_k generated by the k th replanning invocation can be approximated by the expression $\rho_k = \rho_{k-1} - 1$. In practice ρ_k may be larger or smaller than ρ_{k-1} . The paths illustrated in Figure 5.14 (c) and (d) show the results of replanning after an error in the execution of the plan shown in Figure 5.14 (b). This illustrates that it is possible for ρ_k to be greater than ρ_{k-1} . If the first segment of the plan in Figure 5.14 (b) was executed in the proper direction but went too far, it is possible that the agent could clear the obstacle and get to the goal. In this case ρ_k would be less than ρ_{k-1} . Using these two assumptions the total macroprocess planning complexity can be expressed as

$$\Pi_{\text{MAC}} = \{(\rho-1) + (\rho-2) + \dots + 1\} * (\kappa*(1+\beta))^b = \Theta(\rho^2).$$

where ρ denotes the number of steps in the original plan. Under the same conditions it is easy to see that the total complexity for RML can be expressed as

$$\begin{aligned} \Pi_{\text{RML}} &= \{\log_b(\rho) + \log_b(\rho - 1) + \dots + 1\} * (\kappa * (1 + \beta))^b \\ &\leq \rho * \log_b(\rho) * (\kappa * (1 + \beta))^b = O(\rho * \log_b(\rho)). \end{aligned}$$

These expressions indicate that the lazy planning nature of the RML architecture results in a significant savings in planning complexity when replanning is an issue.

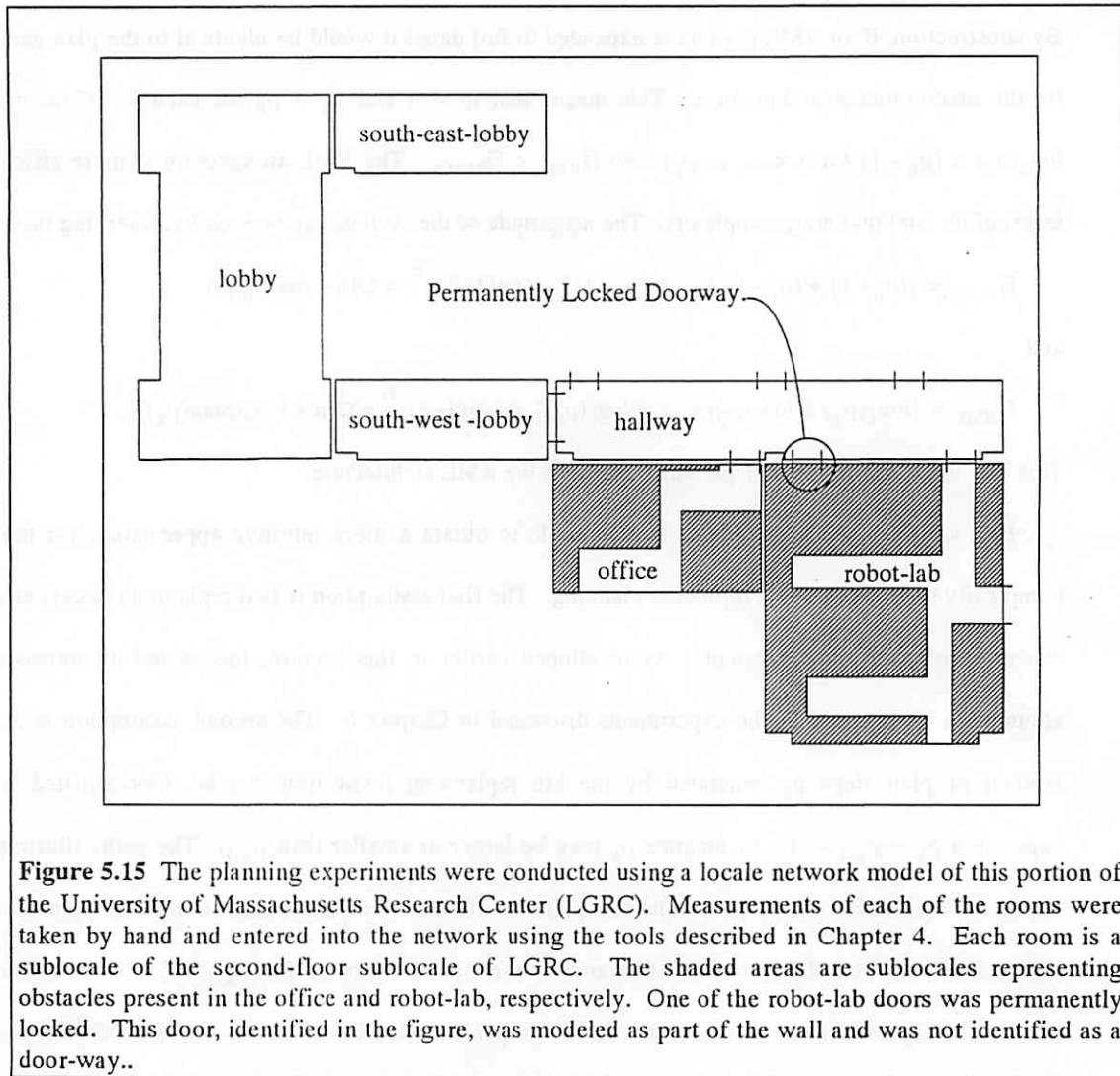


Figure 5.15 The planning experiments were conducted using a locale network model of this portion of the University of Massachusetts Research Center (LGRC). Measurements of each of the rooms were taken by hand and entered into the network using the tools described in Chapter 4. Each room is a sublocale of the second-floor sublocale of LGRC. The shaded areas are sublocales representing obstacles present in the office and robot-lab, respectively. One of the robot-lab doors was permanently locked. This door, identified in the figure, was modeled as part of the wall and was not identified as a door-way..

In addition to the lazy planning aspect of the RML architecture, perception is used extensively to keep the vehicle executing actions accurately and according to plan. As mentioned earlier in this section, Chapter 6 shows that perception guided execution is capable of keeping errors quite small, making it probable that plan sketches can be used without being reconstructed. Execution of the plan of Figure 5.12

(b), for example, would result in only minor perturbations. This tends to bring the planning back to the lazy planning ideal, bringing the complexity back to $O(\rho)$.

In summary, in a static environment and in the absence of execution errors both macroprocess and RML planning can be done in $O(\rho)$ operations. In environments containing surprises or in which errors in action execution are produced, planning complexity will grow to $O(\rho^2)$ for the macroprocess case. However, the lazy planning characteristic evident in RML keeps this complexity to $O(\rho * \log_b(\rho))$ and the frequent use of vision tends to reduce the complexity to $O(\rho)$. This supports the claim that the RML architecture makes planning efficient. These arguments were based on a worst case analysis, however. Two questions remain: How efficient is the planning in "typical" situations? and How reasonable are the final paths? These questions are discussed in section 5.3.

5.3 Experimental Results

To get some experience with "typical" performance and the "reasonableness" of resulting plans, several experiments have been run using plan-and-monitor to generate paths on the second floor of the University of Massachusetts Research Center. Since the purpose of these experiments was only to investigate planning, they were run without the robot. The execution of primitive actions and the recognition of milestones were assumed to be accomplished successfully. Experiments relating to primitive actions, and milestone recognition will be described in Chapter 6. The following discussion and results concern the performance of the plan-and-monitor algorithm and compare its complexity and execution time with that of the A* algorithm without the focusing mechanisms.

Figure 5.15 shows a *floorplan* of the portion of the model used in these experiments. The data for this area was collected by hand, using a steel tape to make careful measurements of the rooms, the objects, and the location of their visual features. This data was entered into the environmental model using the "compiler" described in Chapter 4. The rooms, lobbies and halls in this model are each sublocales of a larger locale bearing the name "second-floor". The environmental network used in the experiments consists of:

- a) An environment locale. This locale was represented with very little detail.

b) The Lederlie Graduate Research Center (LGRC) locale, which was represented as a rectangular volume with no visual detail and very little detail about the physical attributes of the volume.

c) The second floor locale, which, like the LGRC locale, was represented with little detail.

d) The sublocales of the second floor locale. These correspond to the rooms labeled in Figure 5.15 as: lobby, South East Lobby, South West Lobby, Hallway, office and robot-lab. These locales were represented with a reasonably high level of detail. Geometric and physical information was represented for walls, pillars, doors, some wall outlets, blackboards and baseboards. Physical information included surface reflectivity and whether or not it was solid (doorway or wall).

e) Sublocales of the office and the robot-lab representing obstacles in those rooms. These were "clutter" locales each representing a collection of tables, chairs, terminals and bookshelves. Only the extent of these cluttered areas was modeled, but these measurements were precise.

The lobbies and hallway are free from obstacles. Most of the locales are very complex, consisting of many faces and color patches. The total number of nodes in this area is 95. This includes the inside planning points, the outside planning points and door-way points associated with the locales.

Each experiment consisted of presenting plan-and-monitor with a goal. These goals were designed to result in a variety of path lengths, but were typical of the kinds of goals a human would formulate when moving about the building. For each experiment the intermediate path results, the final path results, the complexity (in terms of the number of nodes expanded) and the elapsed time were recorded. The intermediate and final paths for two of those experiments are plotted in Figures 5.16-5.25. Sections 5.3.1 and 5.3.2 describe the details of two of the experiments. Section 5.3.3 discusses and summarizes the results.

5.3.1 Experiment 1

The first experiment presents plan-and-monitor the goal of moving from Harvey's home location, a point in the robot laboratory, to a point just inside Allen's office. More precisely the goal to be accomplished is:

(ptrans robot-lab start-point office goal-point)

The robot-lab start-point and office goal-point locations are identified in Figure 5.16. The first invocation of sketch-a-plan expanded this goal into a plan sketch consisting of five subgoals:

(ptrans robot-lab start-point robot-lab door-1)
(mtrans robot-lab door-1 hallway door-1)
(ptrans hallway door-1 hallway door-2)
(mtrans hallway door-2 office door-2)
(ptrans office door-2 office goal-point)

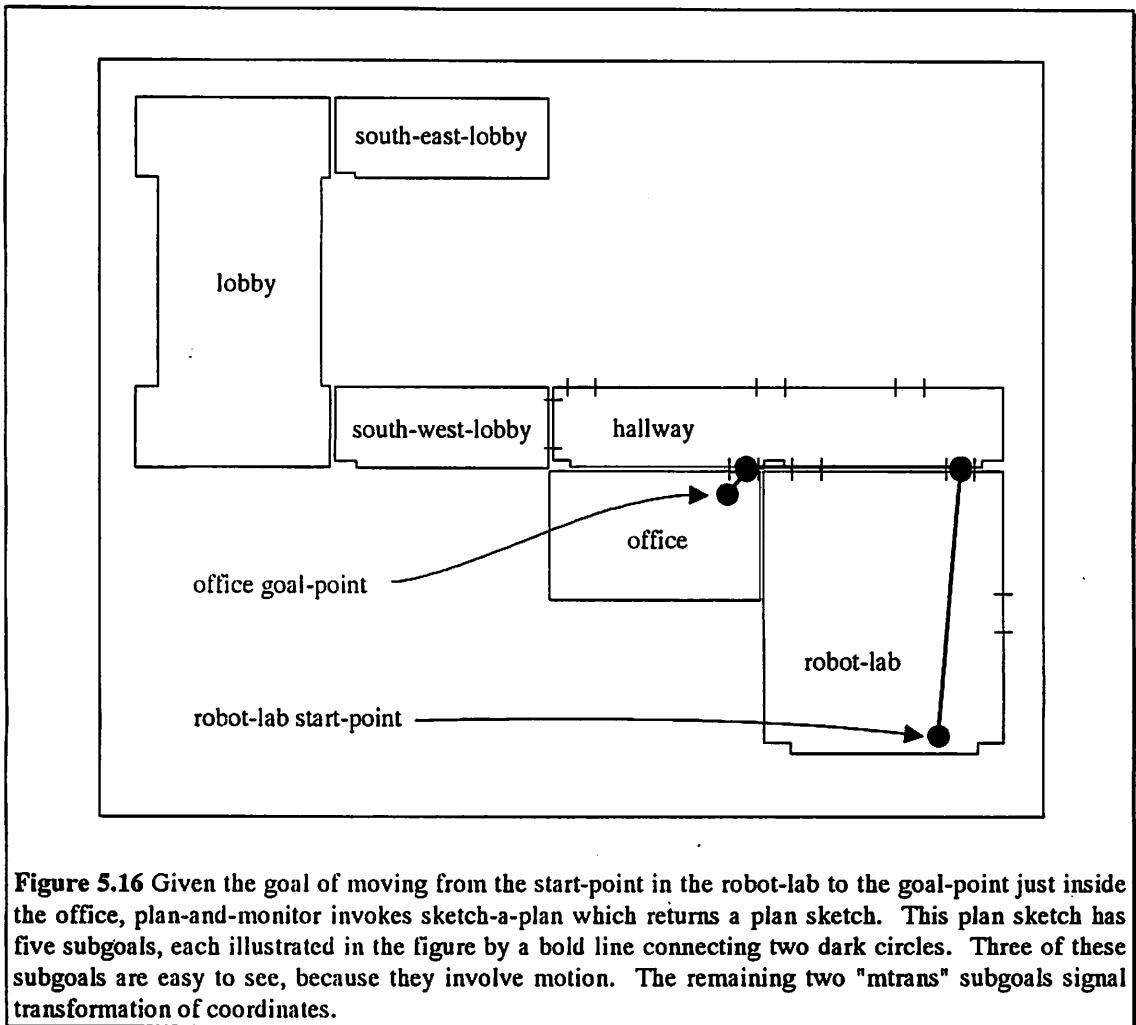


Figure 5.16 Given the goal of moving from the start-point in the robot-lab to the goal-point just inside the office, plan-and-monitor invokes sketch-a-plan which returns a plan sketch. This plan sketch has five subgoals, each illustrated in the figure by a bold line connecting two dark circles. Three of these subgoals are easy to see, because they involve motion. The remaining two "mtrans" subgoals signal transformation of coordinates.

Each of these subgoals is represented in Figure 5.16 as a bold straight line connecting two dark circles. The dark circles represent the start and end locations for the goal. The two mtrans goals in this set are primitive, since they involve only a transformation of coordinates. These subgoals are difficult to see in the figure because start and finish locations in mtrans subgoals are usually identical. The other subgoals in the plan sketch are not primitive. Accomplishing them requires avoiding obstacles: the cluttered areas of the office and robot-lab and pillars in the hallway wall.

Plan-and-monitor then determined that the first of these subgoals:

(ptrans robot-lab start-point robot-lab door-1)

is not primitive and invoked sketch-a-plan to refine it. The result was the development of four new subgoals. These new subgoals, which became the leading subgoals in the plan sketch, become the plan

for moving to the robot-lab door (as shown in Figure 5.17). In the figure it appears that the first two of these subgoals could be collapsed into one. Close inspection of the data, however, showed that the two subgoals were necessary to avoid the obstacle corner identified in the figure. In this new plan sketch the first four subgoals are primitive and would get Harvey to the door without hitting any obstacles. The path to the doorway is optimal. It is exactly the same as the one generated by the A* program for this portion of the trip.

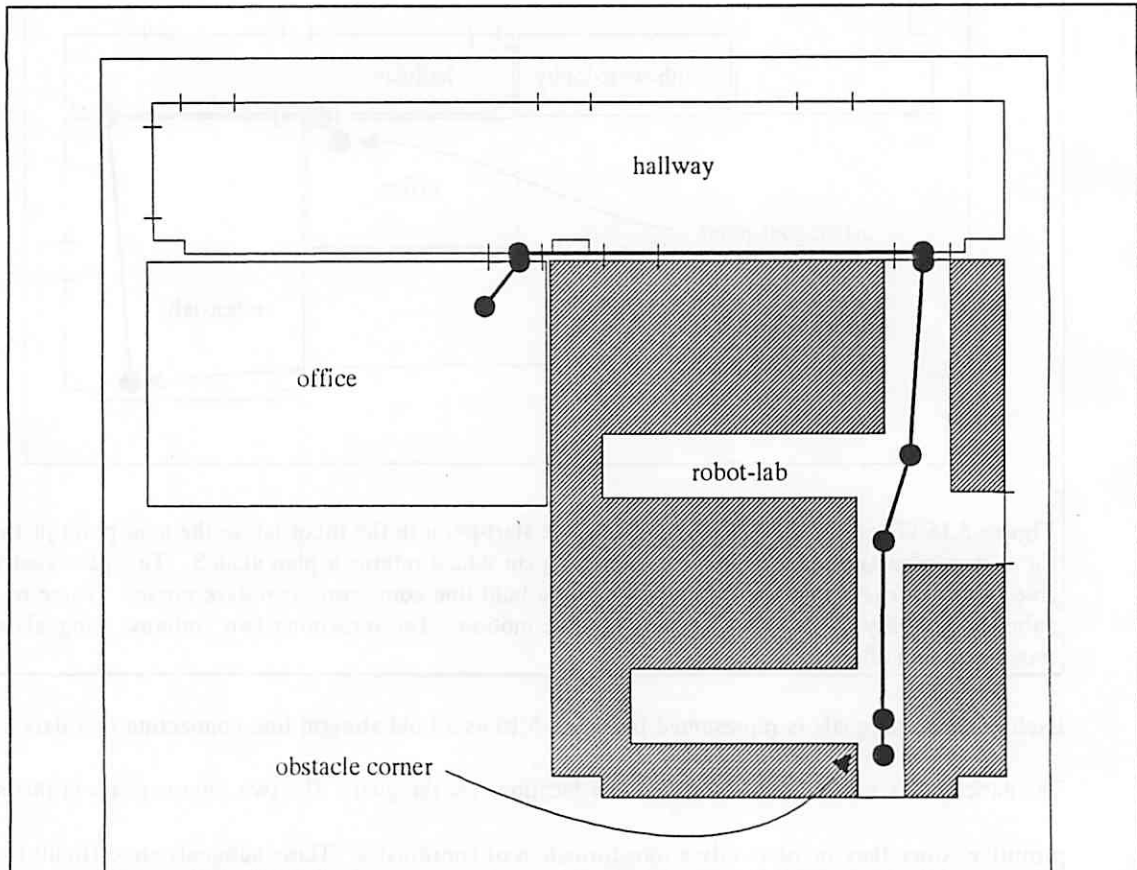


Figure 5.17 The first subgoal of figure 5.16 was not primitive because objects are in the way. So plan-and-monitor refines it by reinvoking sketch-a-plan. The resulting path in the robot lab is made up of four primitive subgoals. Motion begins.

At this point motion begins and under the assumptions of perfect execution, the first four subgoals are achieved. The agent arrives at the door. The next subgoal in the plan sketch is the mtrans expression (mtrans robot-lab door-1 hallway door-1).

This is a primitive subgoal. It signals the transformation of coordinate systems from the robot-lab locale to the hallway locale. The agent also moves 6 inches to traverse the distance between the locales corresponding to the thickness of the wall. The next subgoal to accomplish was:

(ptrans hallway door-1 hallway door-2)

As is clear from Figure 5.17, this was not primitive. To accomplish this subgoal it is necessary to clear the doorway and avoid the hallway pillars. Sketch-a-plan was invoked to refine this goal. Figure 5.18 shows the resulting subgoals for the trip through the hallway. This path clears the doorway from the robot lab to the office and avoids the pillar near the doorway from the hallway to the office. All the subgoals in this subpath were primitive and the subpath is optimal.

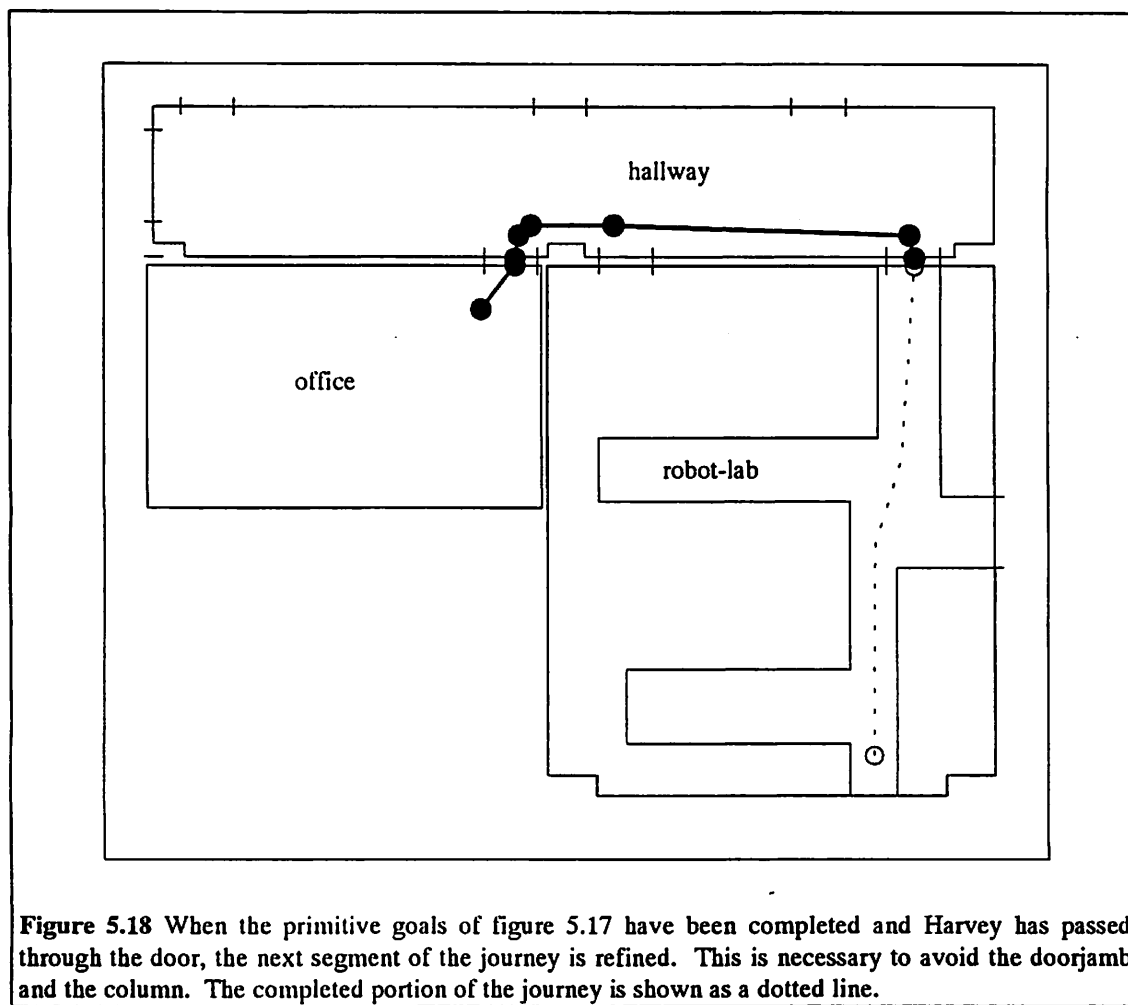


Figure 5.18 When the primitive goals of figure 5.17 have been completed and Harvey has passed through the door, the next segment of the journey is refined. This is necessary to avoid the doorjamb and the column. The completed portion of the journey is shown as a dotted line.

Once these five new subgoals were accomplished and the robot "arrived" at the door to the office, the next mtrans expression:

(mtrans hallway door-2 office door-2)

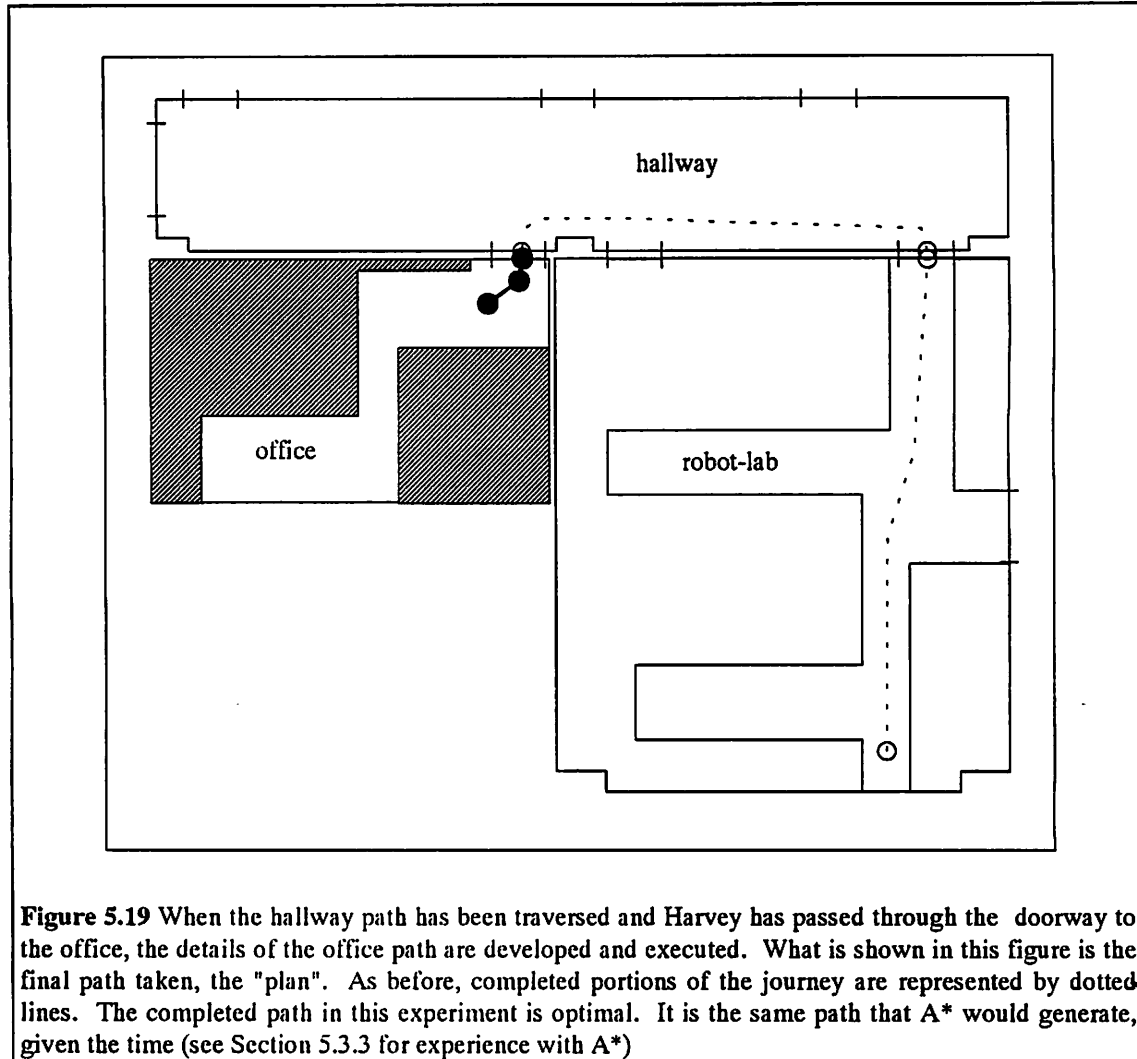


Figure 5.19 When the hallway path has been traversed and Harvey has passed through the doorway to the office, the details of the office path are developed and executed. What is shown in this figure is the final path taken, the "plan". As before, completed portions of the journey are represented by dotted-lines. The completed path in this experiment is optimal. It is the same path that A* would generate, given the time (see Section 5.3.3 for experience with A*)

moved the agent through the door and invoked a transformation of coordinate systems, this time to the local system for the office.

This brings execution of the plan sketch to the final subgoal

(ptrans office door-2 office goal-point)

which is not primitive, only because it was necessary to clear the doorway. A straight line path from the doorway to the goal point would result in a collision with the door jam. The plan in the figure avoided

this by moving into the room before heading for the goal. This subgoal was refined as shown in Figure 5.19. The complete path generated from the start point in the robot-lab to the goal-point in the office for this experiment was both locally and globally optimal. It is exactly the path which would be generated by A*. It is the shortest path which will avoid all the obstacles. It is certainly "reasonable".

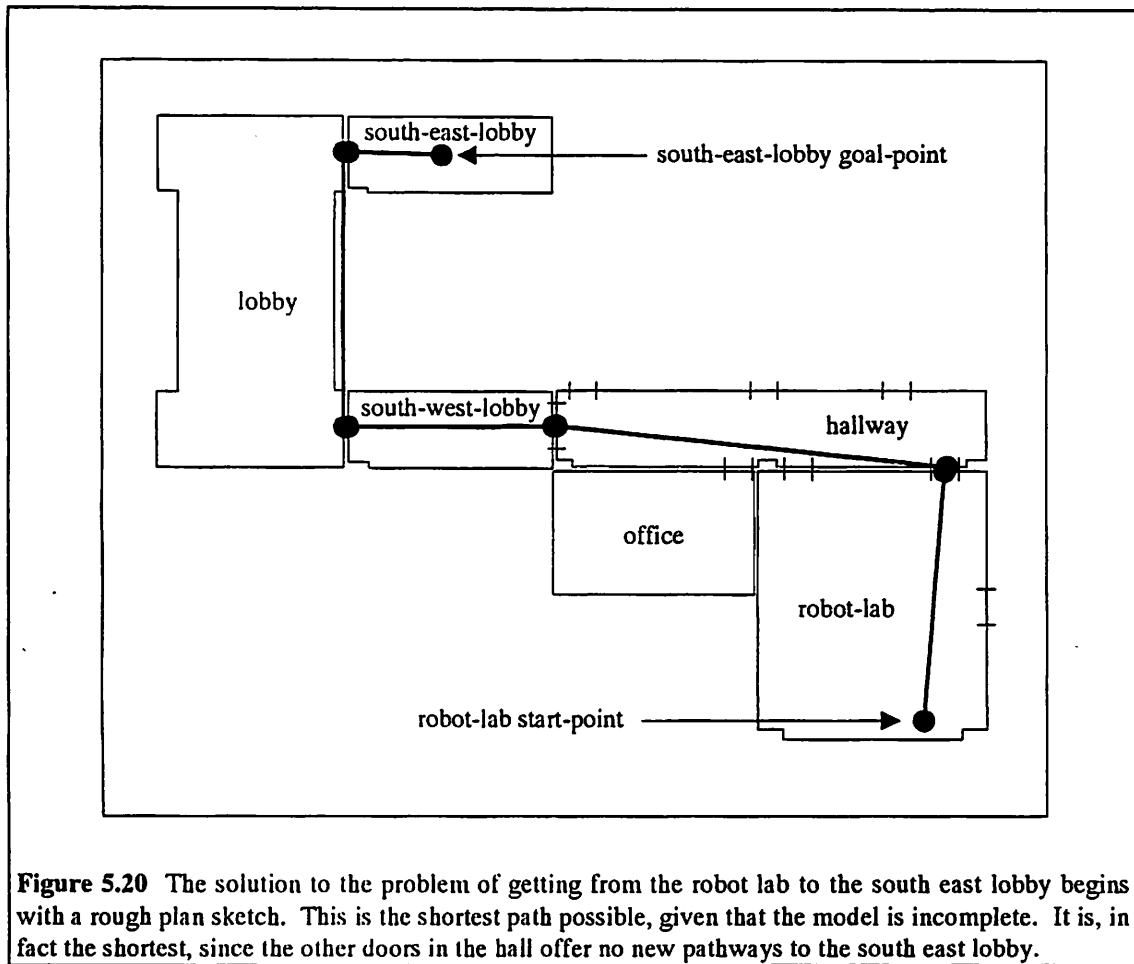


Figure 5.20 The solution to the problem of getting from the robot lab to the south east lobby begins with a rough plan sketch. This is the shortest path possible, given that the model is incomplete. It is, in fact the shortest, since the other doors in the hall offer no new pathways to the south east lobby.

5.3.2 Experiment 2

In the experiment described in section 5.3.1 the final path was optimal. Remembering the discussion in Section 5.2.2 comparing the path lengths produced by A* and plan-and-monitor, this result is not surprising. The doors encountered in this route were narrow and the experiment did not exercise any of the effects illustrated in Figure 5.12. The second experiment was designed to reveal some of these effects. This time plan-and-monitor was presented with a goal designed to result in a path which would

traverse more and varied locale boundaries . The goal was to move from Harvey's "home position" in the robot lab to a point in the south-east-lobby. The goal to be accomplished was:

(ptrans robot-lab start-point south-east-lobby goal-point)

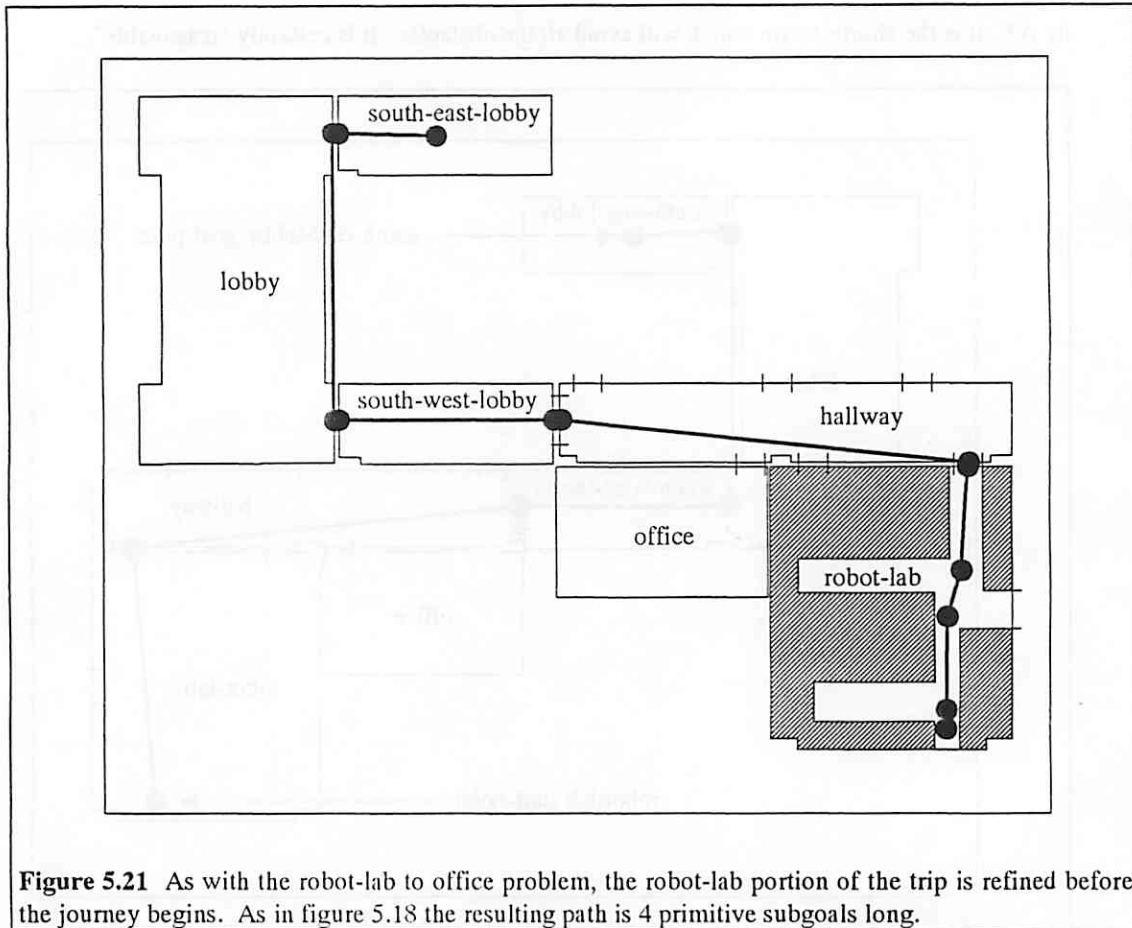


Figure 5.21 As with the robot-lab to office problem, the robot-lab portion of the trip is refined before the journey begins. As in figure 5.18 the resulting path is 4 primitive subgoals long.

Figure 5.20 identifies the start and finish locations of this goal and shows the plan sketch resulting from the first invocation of sketch-a-plan. This plan sketch consists of the following 9 subgoals:

- (ptrans robot-lab start-point robot-lab door)
- (mtrans robot-lab door-1 hallway door-1)
- (ptrans hallway door-1 hallway-door-2)
- (mtrans hallway door-2 south-west-lobby door-1)
- (ptrans south-west-lobby door-1 south-west-lobby door-2)
- (mtrans south-west-lobby door-2 lobby door-1)
- (ptrans lobby door-1 lobby door-2)
- (mtrans lobby door-2 south-east-lobby door-2)
- (ptrans south-east-lobby door-2 south-east-lobby goal-point)

During the execution of this plan sketch the agent encounters four locale transitions. Two of them are doorways: the first is the narrow door between the robot-lab and the hallway; the second is the wider door opening from the hallway into the south-west-lobby. The other two transitions are locale divisions of the type illustrated in Figure 5.13 (e). These divisions result from dividing a large locale into sublocales by creating artificial door-way faces where the agent is likely to change course anyway.

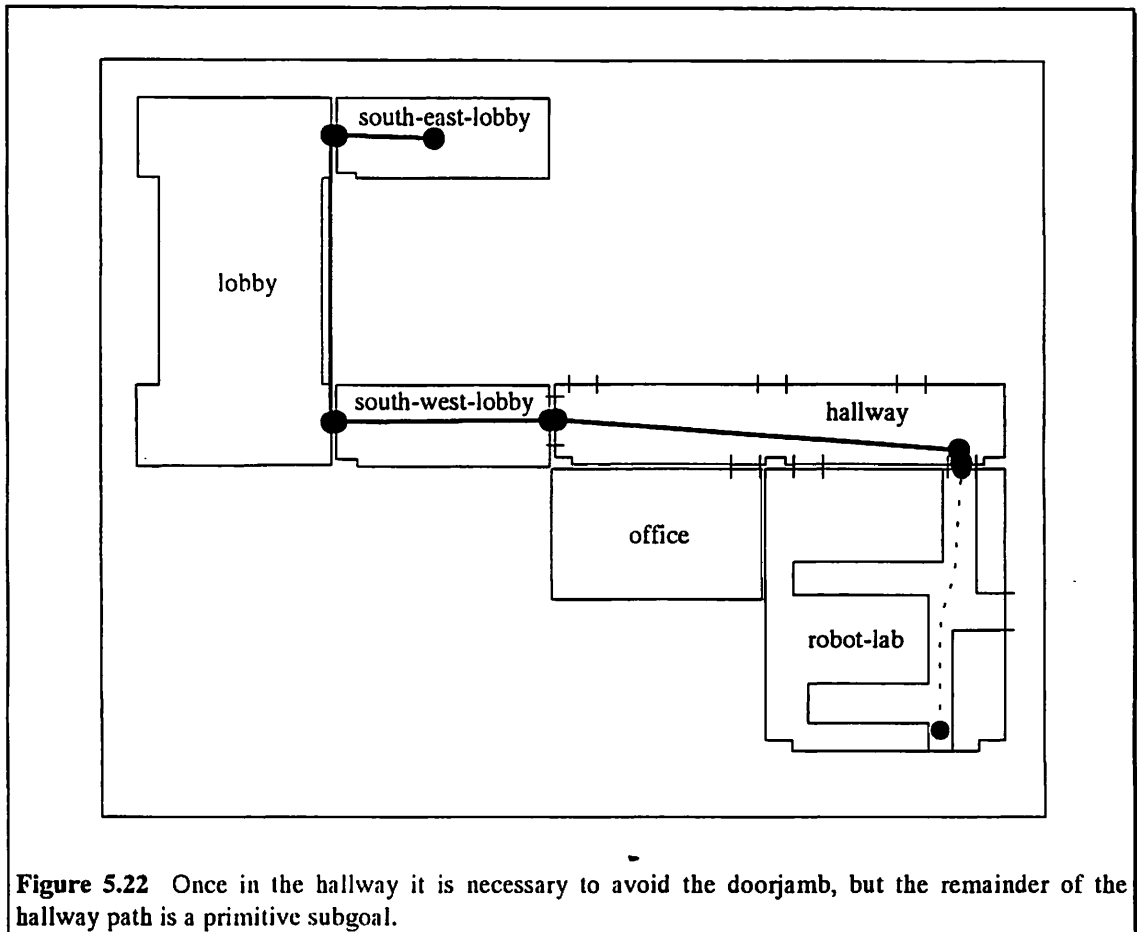


Figure 5.22 Once in the hallway it is necessary to avoid the doorjamb, but the remainder of the hallway path is a primitive subgoal.

The first two subgoals in this plan sketch are identical to the corresponding subgoals generated in the first experiment. The result is the same, as can be seen by comparing Figures 5.18 and 5.21. This brings the agent into the hallway. The next subgoal

(ptrans hallway door-1 hallway-door-2)

had to be refined. The resulting path, shown in Figure 5.22, consisted of two segments: a very short one, to clear the doorway, and a long one, to move to the next doorway.

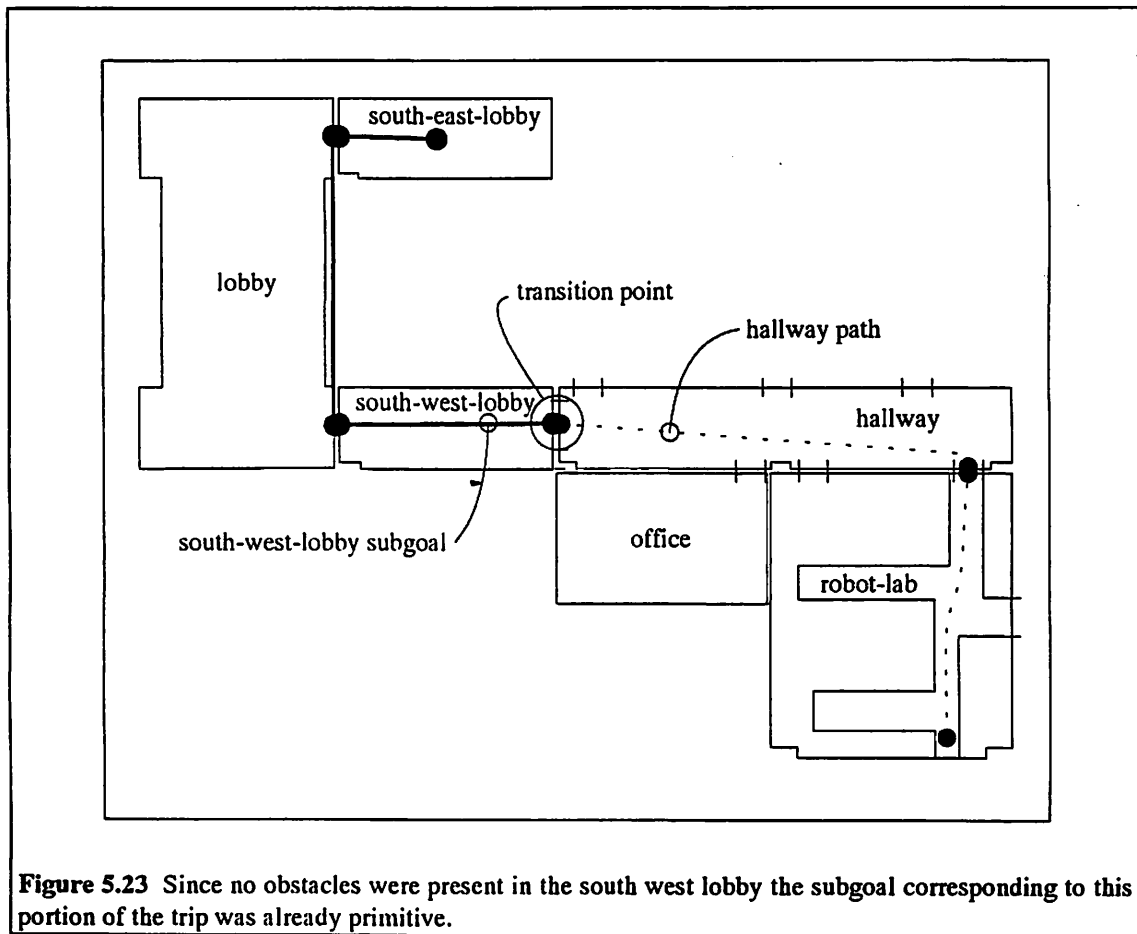
The next subgoal

(mtrans hallway door-2 south-west-lobby door-1)

caused the agent to move through the doorway and change coordinate systems. Now, reasoning about the south-west-lobby locale, the subgoal

(ptrans south-west-lobby door-1 south-west-lobby door-2)

was found to be primitive. There are no obstacles in the locale and the doorways are both very wide. The subgoal is shown in Figure 5.23. It also becomes clear in Figure 5.23 that the location of the transition from the hallway to the south-west-lobby is not optimal. This doorway is quite wide, so it would have been physically possible to create a shorter path to accomplish what is done by the combined hallway path and south-west lobby subgoal. That "ideal" A* path would have no fewer path segments, however. The door jam would still need to be avoided.



Once again the system makes a translation of coordinate systems:

(mtrans south-west-lobby door-2 lobby door-1),

crossing into the lobby. The details of the lobby locale make it clear that accomplishing the subgoal:

(ptrans lobby door-1 lobby door-2)

requires that the agent avoid the lobby pillars. This results in the refinement shown in Figure 5.24. This figure makes it clear that location of the transition point from the south-west-lobby to the lobby is not optimal. A shorter path is possible and the transition point creates an extra path segment. The number of path segments is increased by one, compared to what A* would produce.

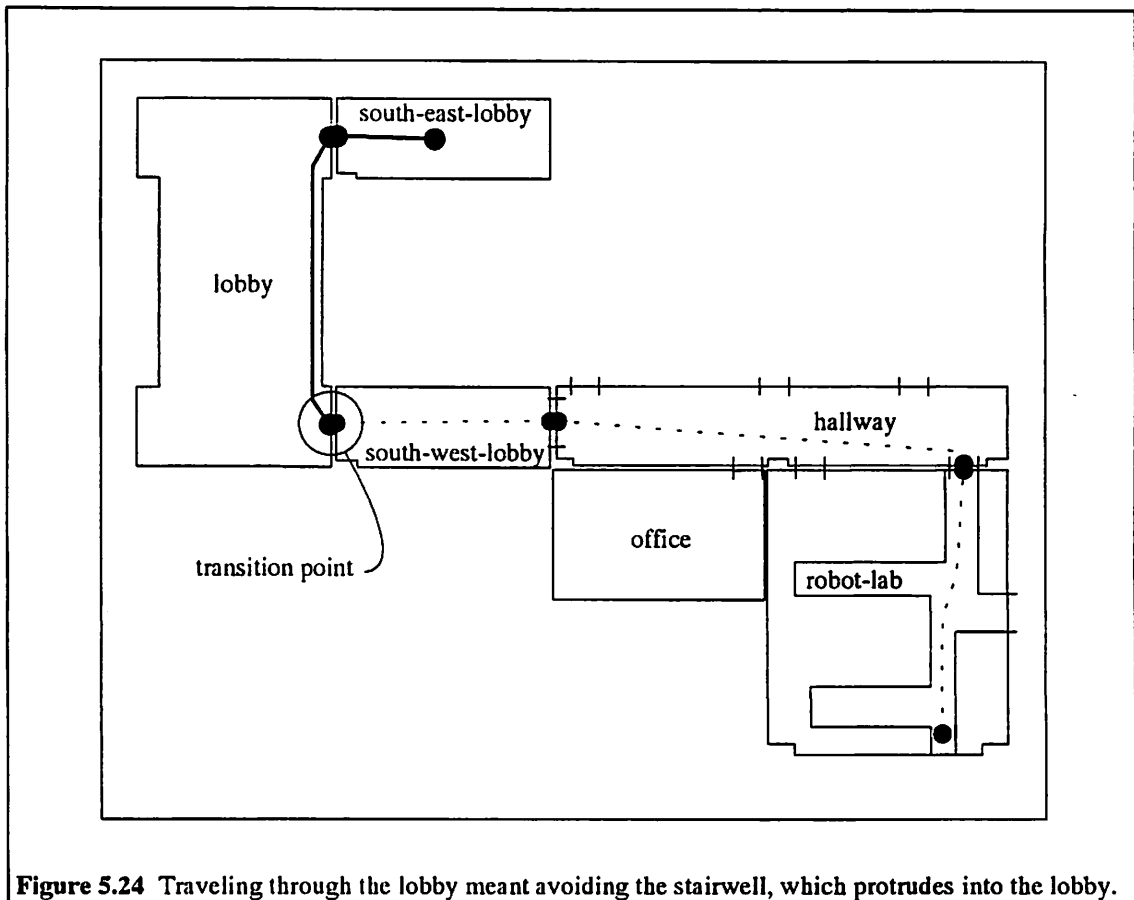


Figure 5.24 Traveling through the lobby meant avoiding the stairwell, which protrudes into the lobby.

What happens in the remainder of the plan sketch:

(mtrans lobby door-2 south-east-lobby door-2)

(ptrans south-east-lobby door-2 south-east-lobby goal-point)

reveals nothing new. The results, however are shown in Figure 5.25.

The final path is "reasonable", but not "ideal". The transition points at three of the doorways make the path longer than it needs to be. In this example, however, the additional distance amounts only to an additional two feet of travel over a distance of about 100 feet. The error is due to the way doorway points are represented, namely as being in the center of the door. There are many ways this can be improved. A simple method would be to follow the design used by [Arkin, 1987]. In his system he represented doorways by three points: one point at the center of the doorway and one point a safe distance from each door jamb. Improving this representation is left as a matter for future research.

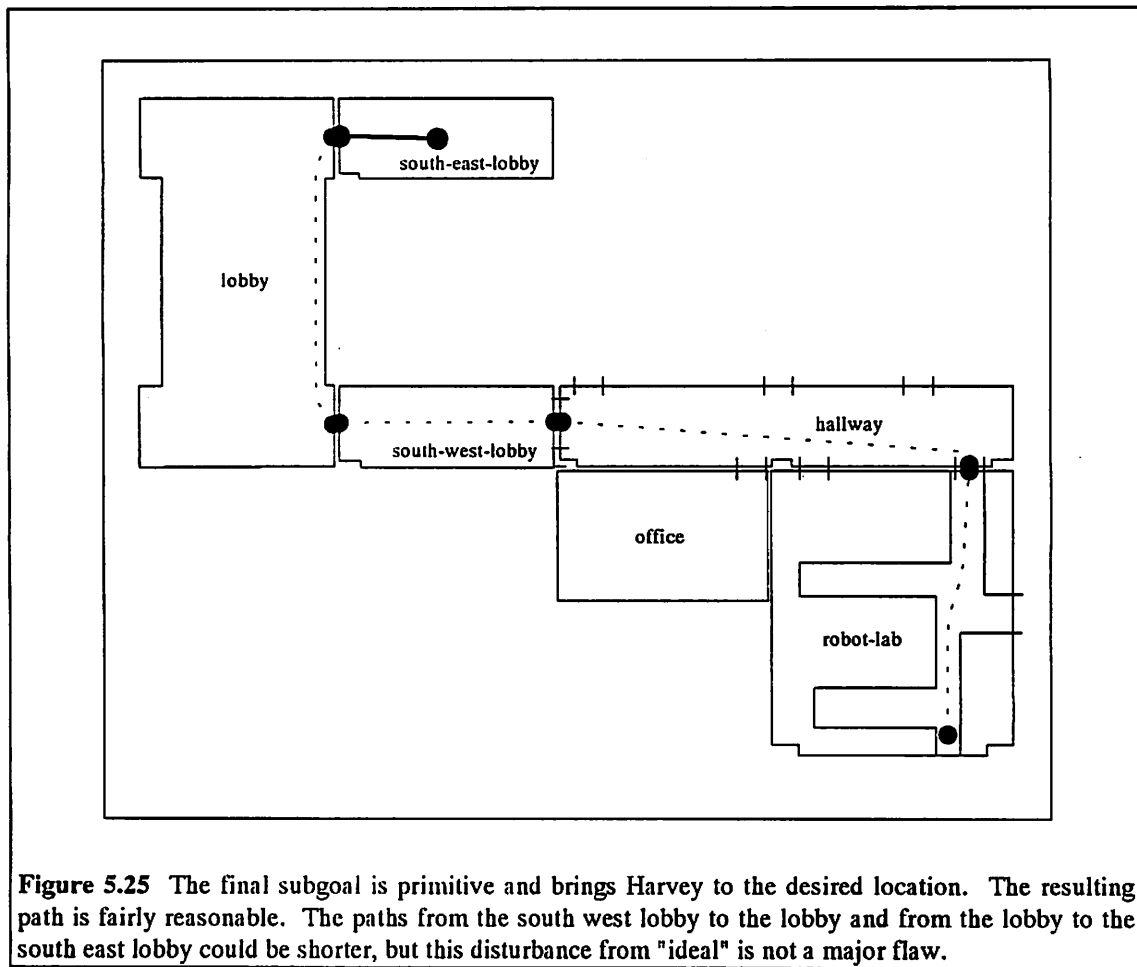


Figure 5.25 The final subgoal is primitive and brings Harvey to the desired location. The resulting path is fairly reasonable. The paths from the south west lobby to the lobby and from the lobby to the south east lobby could be shorter, but this disturbance from "ideal" is not a major flaw.

These experiments show that, although final paths are not always optimal, they are "reasonable". The departure from optimality in terms of distance traveled is only a few percent in these experiments and difference in the number of path segments is small. Pathological cases can be constructed to make

these differences larger, but the experiments illustrate "typical" situations. In the next section the complexity of planning computations are determined experimentally for these "typical situations".

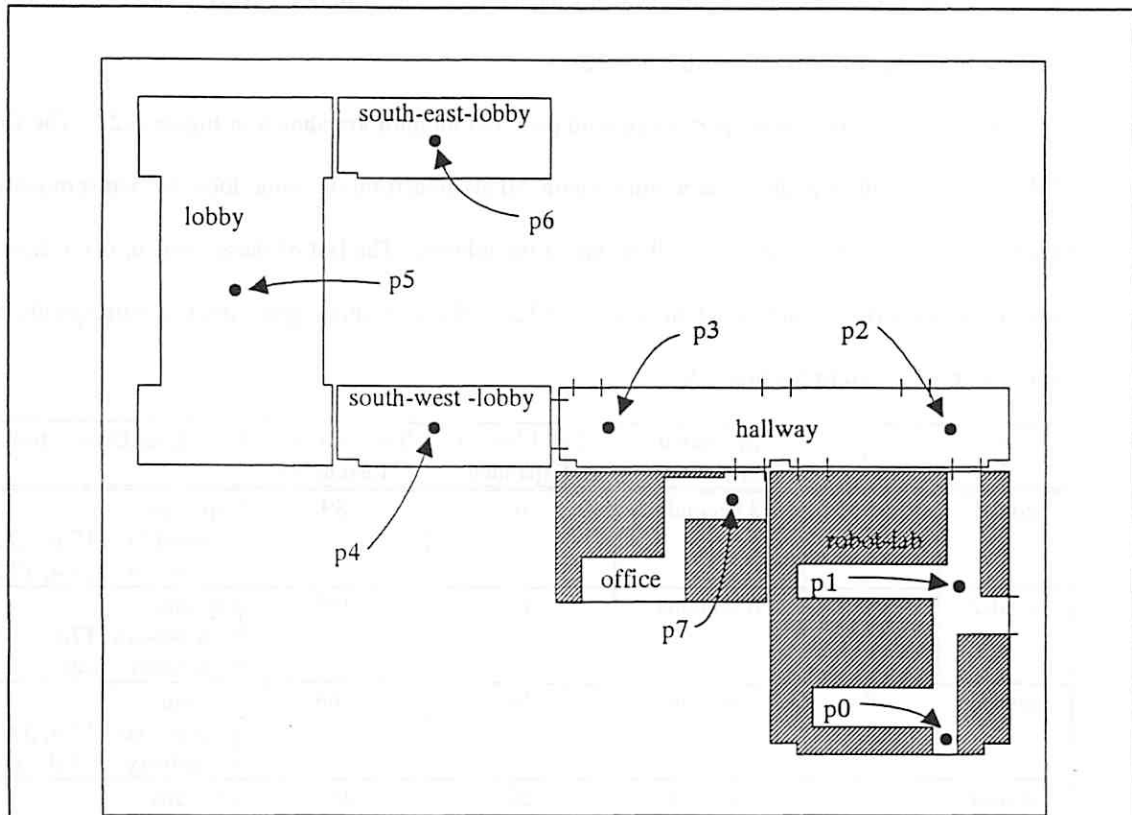


Figure 5.26 The goals used in the complexity experiments described in section 5.3.3 are ptrans expressions using the locations p_k identified above. For each k , goal- k refers to the goal of moving from p_0 to p_k . Goal 5, for example is (ptrans robot-lab p_0 lobby p_5).

5.3.3 Experimental Experience with Planning Complexity in "Typical" Situations

The complexity analysis of section 5.2 claimed that using the plan-and-monitor algorithm results in dramatic improvements in planning complexity over an algorithm such as A*. These claims, however, were based on mathematical arguments founded on a worst case analysis. In order to get some experience with how these algorithms behave in "typical" situations, a second set of experiments were performed. Each experiment consisted of presenting a goal to one of the planners and recording the cpu time used and the number of nodes expanded. The goals were chosen to be typical of the kinds of goals which would generated in a "go fetch" problem and to result in paths of a variety of lengths. The specific goals chosen, labeled goal-1, goal-2, ... , goal-7, are illustrated in Figure 5.26. Each goal, goal- k refers to

a ptrans expression with p0 as its start location and pk as its finish location. The goal, goal-7, for example, corresponds to the expression (pttrans robot-lab p0 office p7). These experiments have been performed on all the machines mentioned in Chapter 1, but the results reported in this section are those obtained from experiments on an AST-486/33.

The results of experiments performed with plan-and monitor are shown in Figure 5.27. The first six of these goals result in paths of increasing length, all starting from the same location, but terminating in various places in the robot lab, the hall or one of the lobbies. The last of these, goal-6, is the same goal used in the experiment described in Section 5.3.2. The remaining goal, goal-7, corresponds to the experiment described in Section 5.3.1.

Goal Label	PM	Measured CPU Time	Total Nodes Expanded	Total Calls To "Reachable"	Goal Description
goal-1	3	3 seconds	6	89	(pttrans robot-lab (17.6, 3.0) robot-lab (19.4, 17.2))
goal-2	8	6 seconds	18	168	(pttrans robot-lab (17.6, 3.0) hallway (40.0, 4.0))
goal-3	8	7 seconds	18	168	(pttrans robot-lab (17.6, 3.0) hallway (5.0, 4.0))
goal-4	11	8 seconds	25	169	(pttrans robot-lab (17.6, 3.0) s-w-lobby (10.0, 4.0))
goal-5	16	10 seconds	41	223	(pttrans robot-lab (17.6, 3.0) lobby (10.0, 18.0))
goal-6	19	17 seconds	55	320	(pttrans robot-lab (17.6, 3.0) s-e-lobby (10.0, 4.0))
goal-7	15	11 seconds	43	252	(pttrans robot-lab (17.6, 3.0) office (18.0 10.5))

Figure 5.27 These results summarize planning complexity experience when the plan-and-monitor algorithm was used. The environment model used in these experiments was the entire UMass model. The experiments were performed on a 33 megahertz Intel 80486 based machine with 10 megabytes of RAM. All code was written in COMMONLISP.

Figure 5.28 displays these results as a graph, plotting the number, N, of nodes expanded as a function of path length. This data is consistent with the complexity discussion of section 5.2.2 which indicated that the total planning effort for plan-and-monitor would be linear in path length.

The A* experiments were run with two different models to show the effects of model size on the complexity of the algorithm. The first set of experiments were run using only the robot-lab portion of the environment used in the plan-and-monitor experiments; in this case, only goal-1 and goal-2 are relevant (all others are outside the robot-lab environment). The relationship between these two environments is clear from the diagram in Figure 5.15. As expected the complexity for A* grew rapidly as a function of path length (Figure 5.29). For goal-1 the results were the same as for plan and monitor. This is to be expected, since the scope and perspective focusing used by plan-and-monitor resulted in the use of A* to solve goal-1 in exactly the same way. A* made more calls to `reachable`, the function which determined whether or not the path between two points is unobstructed, than did plan-and-monitor. This is also to be expected because the semantic filtering used by plan-and-monitor removed the outside planning points of the robot-lab from consideration. A* does not. The result for goal-2 already shows considerable growth in the number of nodes expanded when compared with plan-and-monitor.

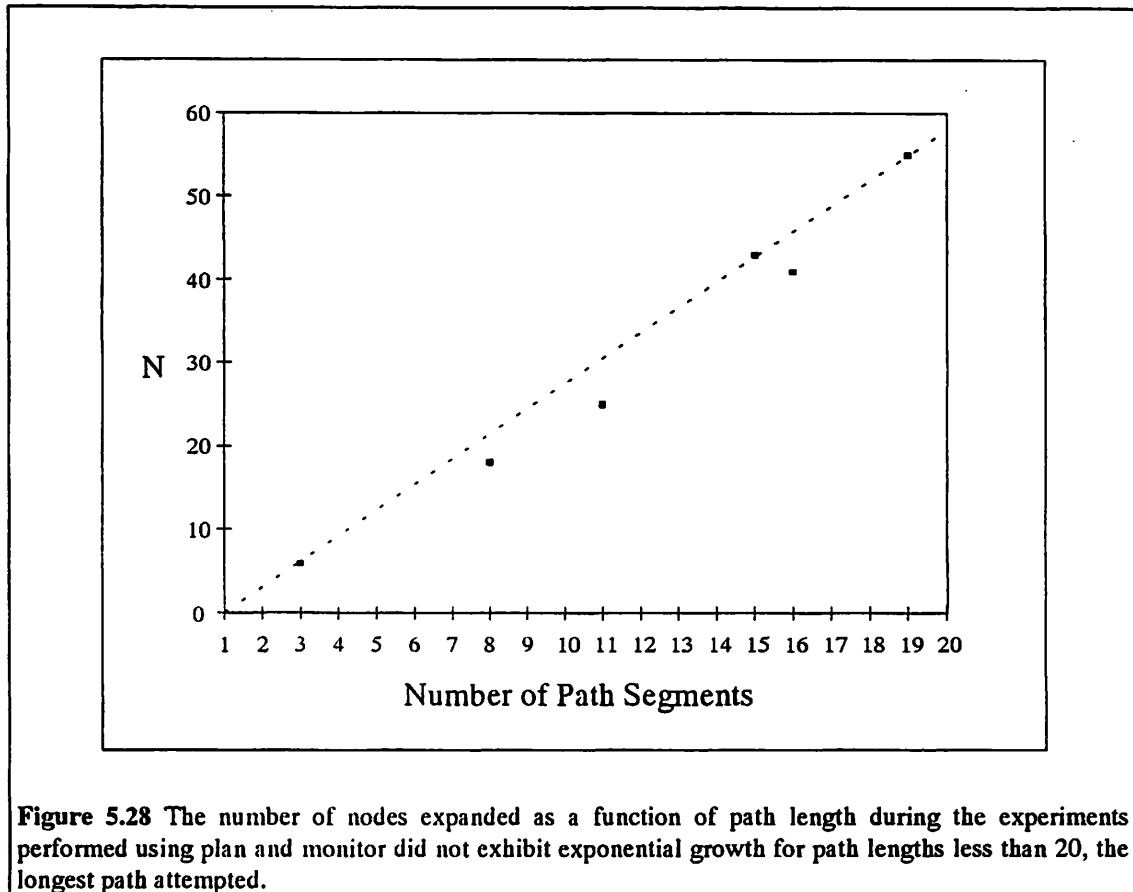


Figure 5.30 shows the results of using A* when the environment is enlarged to include the entire model of Figure 5.15. Again, as expected, the number of nodes expanded climbed rapidly with path length. This growth was so rapid that goals-3, ... , goal-7 were omitted from the experiments. The experiment on goal-3 was terminated when the elapsed time for the run reached 3 hours since this was not a study of the performance characteristics of A*. The point of the experiments had already been made and collecting astronomical data was of dubious value.

Goal Label	ρ_A	Measured CPU Time	Total Nodes Expanded	Total Calls To "Reachable"	Goal Description
goal-1	3	3 seconds	13	146	(ptrans robot-lab (17.6, 3.0) robot-lab (19.4, 17.2))
goal-2	8	14 seconds	77	626	(ptrans robot-lab (17.6, 3.0) hallway (40.0, 4.0)

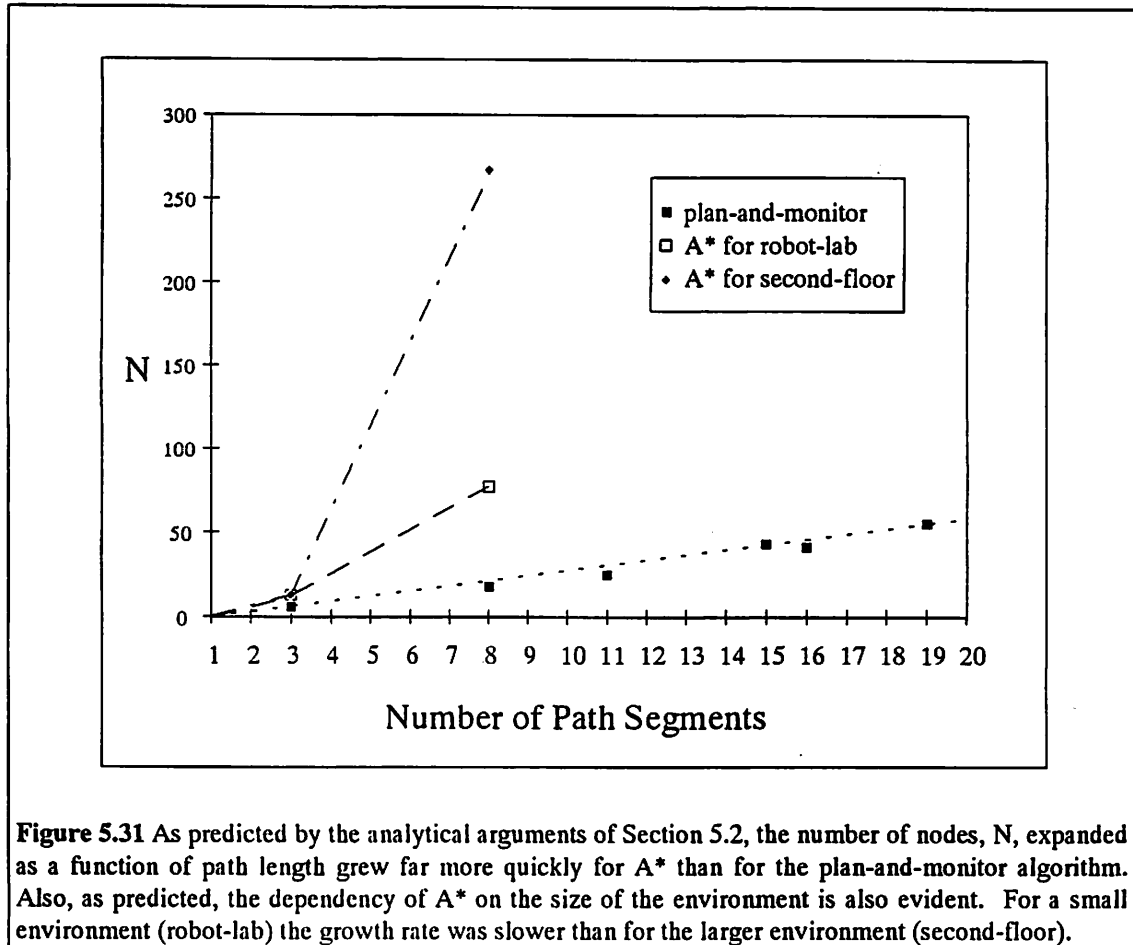
Figure 5.29 When planning was done using the A* algorithm using only the model for the robot-lab the results were quite reasonable, but the measured complexity was higher. The code used in these experiments was written in COMMONLISP and executed on an Intel 80486 based machine with 10 megabytes of RAM.

Goal Label	ρ_A	Measured CPU Time	Total Nodes Expanded	Total Calls To "Reachable"	Goal Description
goal-1	3	90 seconds	13	785	(ptrans robot-lab (17.6, 3.0) robot-lab (19.4, 17.2))
goal-2	8	1860 seconds	267	10181	(ptrans robot-lab (17.6, 3.0) hallway (40.0, 4.0)

Figure 5.30 Expanding the model to include the second-floor of the Research building increased the measured complexity for the A* algorithm considerably. Execution times grew disproportionately due to an increased amount of disk activity. The code used in these experiments was written in COMMONLISP and executed on an Intel 80486 based machine with 10 megabytes of RAM.

A comparison of Figure 5.29 and Figure 5.30 shows the effect of increasing the size of the model (the number of nodes). This consistent with the $O(\eta^\rho)$ growth characteristics of A* as discussed in Section 5.14. Note that, since there are more nodes to consider, there is growth of the number of calls to the function "reachable". This is clearly a major contributor to the CPU time in the results presented in

Figure 5.30. It is not, however, a surprise complexity issue. For each node expanded the successor function is called. This function calls "reachable" once for each node in the model. For a given model the number of calls to reachable is proportional to the number of nodes expanded.



The A* results are combined with those for plan-and-monitor in Figure 5.31. These results indicate that the complexity claims revealed by the analytical arguments of section 5.2 have a very real effect in typical situations.

CHAPTER 6

PERCEPTION AND THE EXECUTION OF PRIMITIVE SUBGOALS

In Chapter 3 the plan-and-monitor algorithm was described as one realization of the RML concept. This algorithm develops plan sketches, using the sketch-a-plan algorithm described in Chapter 5, by decomposing subgoals in a depth first fashion until the next subgoal to be accomplished is a primitive one. It is at this point that the plan-level servo loop moves from its "reason a little" increment and begins to "move a little". Sufficient reasoning has been done to formulate the next primitive subgoal; the next step is to attempt to accomplish that subgoal. On completion of the "move-a-little" step, plan-and-monitor will "look-a-little" to check if reasoning and reality are in agreement. The move-a-little step in the plan-level servo loop involves additional reasoning about motion and perception. This chapter describes that reasoning, shows that the reasoning about motion and vision are simplified by the fine-grained nature of the RML architecture and demonstrates by experiment that the resulting motions are accurate enough to clearly justify the arguments of Section 5.2.3.

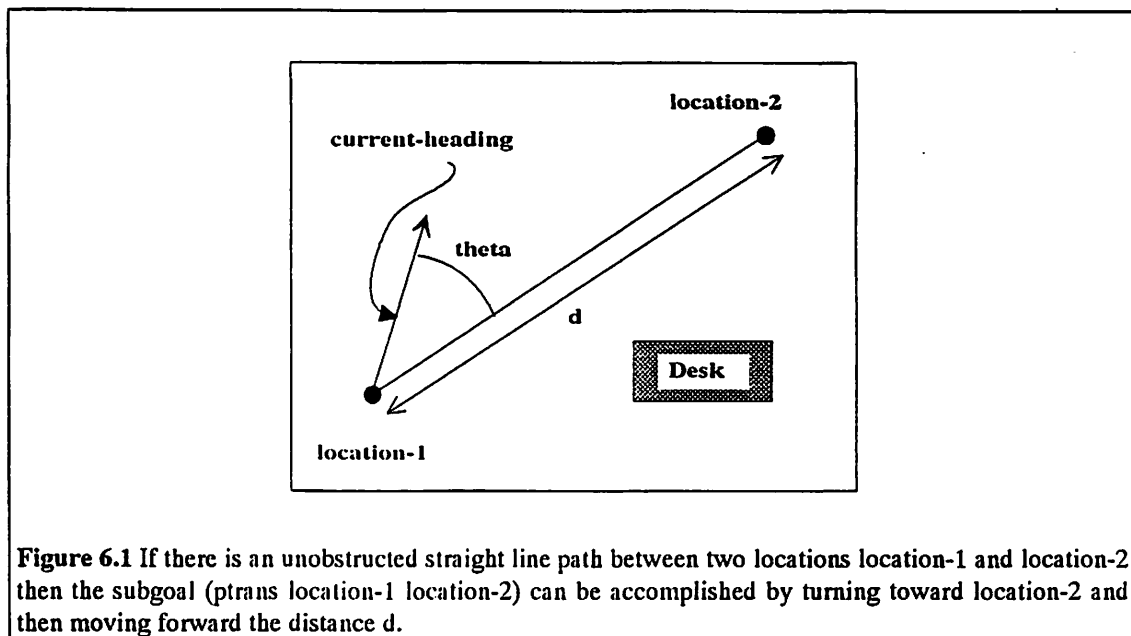


Figure 6.1 If there is an unobstructed straight line path between two locations location-1 and location-2 then the subgoal (ptans location-1 location-2) can be accomplished by turning toward location-2 and then moving forward the distance d.

This chapter begins in section 6.1 with a discussion of the concept of a primitive subgoal, indicating the relationship between primitive subgoals and primitive actions. Section 6.2 describes the mechanisms

used to accomplish these primitive subgoals and to verify that they have been attained. This discussion will follow the perceptual servoing analogy described in section 3.2.3 of chapter 3, showing how plan-level and action-level servoing are implemented in the experimental system. Experimental results in this section support the need in Chapter 5 for execution accuracy as a means to keep planning computation low. Section 6.3 analyzes the complexity of the two implemented servo loops and sketches some ideas for goal-level servoing which would be simplified by the RML architecture. Finally, Section 6.4 summarizes the effects of interweaving reason, action and perception.

6.1 Primitive Subgoals

A primitive subgoal, as defined in chapter 3, is one which can be directly executed without further subgoal refinement. This definition explains the role of primitive subgoals in the planning process, but it does not relate the concept to the capabilities of the agent. What is or is not a primitive subgoal depends on the capabilities of the agent, and may change in a given agent through the process of learning. An example primitive subgoal goal for a human agent might be captured in the English statement:

"Drive down the road until you get to the third stoplight."

Most adult humans are able accomplish this subgoal with little effort and no plan development, but for a simpler agent, such as Harvey, this could be a formidable task. Harvey's set of primitive actions are modest, consisting only of (turn angle) and (move distance) and he is not equipped with the complex scripts or behaviors necessary to orchestrate his sensors and these primitive actions into such a complex behavior. Traversing an unobstructed straight line path between two points is a behavior more in line with Harvey's capabilities. In principle it is always possible to accomplish such a subgoal by first turning toward the goal location and then moving the required distance (Figure 6.1). Thus, one type of primitive subgoal for Harvey would be a goal, (pttrans location-1 location-2), which satisfies the condition that the path from location-1 to location-2 is unobstructed. No further planning is required for a subgoal of this type because such a subgoal could, in principle, be satisfied by executing the script:

```
(script-pttrans-clearpath (location-1 location-2)
  (turn (- (angle location-1 location-2)) heading)
  (move (distance location-1 location-2)))
```

This script specifies that to get from location-1 to location-2 it is only necessary to perform two actions: to first turn the difference between the angle of the line from location-1 to location-2 and the current robot heading, then move the distance between these two locations.

This concept is used as a definition of Harvey's primitive subgoal. It highlights the relationship between primitive subgoals and primitive actions. The script, however, will only satisfy the subgoal if the individual actions "turn" and "move" are properly executed. In practice, this is unlikely without the use of sensory feedback because the realities of the environment and the agent's actuators make the execution of open loop actions unreliable. Whenever Harvey performs "turn", for example, slippage of the rubber tires on the surface introduces a translational motion or "skittering". Moreover, as Harvey moves forward he may be thrown off course by a slippery spot, an unevenness of the surface, or even a bulge in one of the tires. To reduce this error the execution of each primitive action is controlled by servoing on prominent visual features in the environment. This servoing using perceptual features is the procedure referred to in chapter 3 as "action-level perceptual servoing".

Action-level perceptual servoing increases the accuracy of each action, but does not relate the result to progress toward the goal, nor does it prevent the accumulation of inaccuracies over a number of such actions. To use driving a car as an example, it may be determined that the stoplights in the above example are 1.23 miles apart. Action level perceptual servoing may make our actions accurate enough to get us close ... say to 1.2 miles (a 2.4% error). After three actions we may be .1 mile short of our goal and will very likely not even be able to see the goal. It is for this reason that milestones have been introduced. A better script would be:

```
(script-pttrans-clearpath (location-1 location-2)
  (turn (- (angle location-1 location-2) heading))
  (if (not(recognize-milestone milestone-t))
      (return (list failure last-known-location)))
  (move (distance location-1 location-2))
  (if (not (recognize-milestone milestone-t))
      (return (list failure last-known-location))
      (return (success location-2))))
```

In this script each action is followed by a milestone check, used to verify that each of the actions has had the planned effect. This is a check to see that reasoning and reality are in agreement. As long as the

results of these checks are positive, processing the script continues, otherwise failure is reported. In any case the last known location is returned along with the success/failure report. If the result of processing the script is (success location-2) the subgoal has been achieved. If, however, the result is (failure last-known-location) then the plan sketch must be either adjusted or redeveloped. If the location last-known-location is sufficiently near location-2, it may be enough to replace the next subgoal to be satisfied, namely (ptrans location-2 location-3), with the new subgoal (ptrans last-known-location location-3) and continue. This procedure is called "plan-level perceptual servoing".

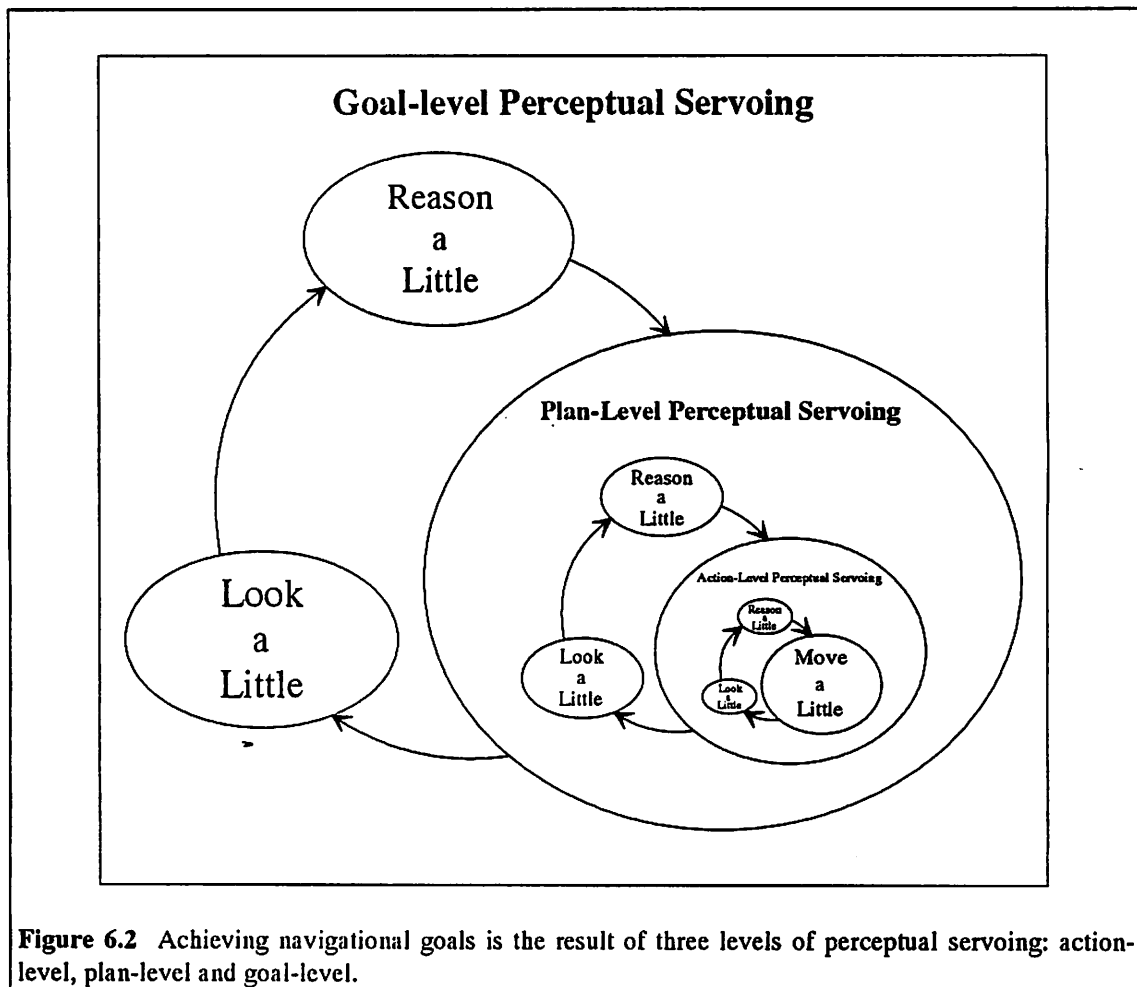


Figure 6.2 Achieving navigational goals is the result of three levels of perceptual servoing: action-level, plan-level and goal-level.

When the distance between the last known location and location-3 is great, however, it will be necessary for the agent to relocate itself and redevelop a major portion of the plan sketch hierarchy. This relocation-redevelopment cycle forms the third level of servoing called "goal-level perceptual servoing". Navigation in the experimental system can be thought of as the result of these three nested perceptual

servoing loops as illustrated in Figure 6.2. Plan-level and Action-level servoing have many design characteristics in common. They are discussed together at length in section 6.2. As has been mentioned, goal level servoing was not implemented as a part of this work. It has been left as a topic for future research.

6.2 Action-Level and Plan-Level Perceptual Servoing

Both action-level and plan-level perceptual servoing are examples of the RML paradigm. The reasoning portion of these perceptual servoing loops analyzes what is known about the environment to determine what action to take next and what should be perceived after that action is complete. The first step of this reasoning is to select the action to be taken and to estimate the location which will result as a consequence of the action. The next step is to select 3D entities, or landmarks, from the model. These landmarks are chosen on the basis of how distinctive they should be and what kind of information they offer the servoing procedure. Once these landmarks are selected their appearance is computed from the model based on the estimate of the agent's next location.

When the reasoning step is complete the action is taken (move a little) and when the action is complete a new image is acquired. The appearance prediction for each landmark is then projected onto the image plane and is matched to data in the image (look a little). The result of this matching, together with the knowledge of the 3D locations of the landmarks, are used in the next reasoning increment to make the appropriate corrections to the action or to the plan sketch hierarchy. Both action-level and plan-level servoing use the same landmark selection, template construction and matching procedures; they differ only in what they do with the resulting information.

6.2.1 Selecting Landmarks

In principle a landmark can be any 3D entity: an object, a group of objects, a group of lines, as described in [Fennema, Hanson, Riseman, Beveridge and Kumar, 1990] and [Beveridge, Weiss and Riseman, 1989, 1990], or a cluster of surface patches, as described here. Surface patches were chosen because their reflectance patterns can be quite distinctive, making it likely that they can be isolated from other 3D entities in an image with rather simple means, such as correlation. Correlation, properly used, is a strong method for matching such reflectance patterns; it is known to be noise tolerant, and it can be

computed quickly on special purpose machinery [Dickmans and Grafe, 1988a,b]. Landmarks are selected on the basis that they will be easy to find using correlation.

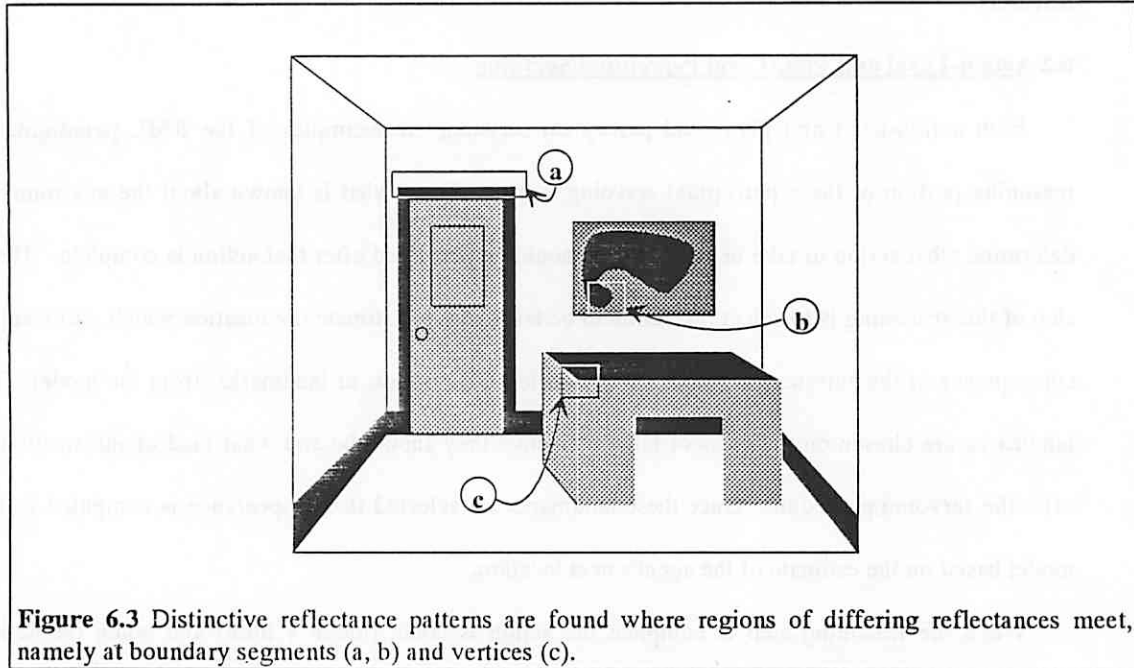


Figure 6.3 Distinctive reflectance patterns are found where regions of differing reflectances meet, namely at boundary segments (a, b) and vertices (c).

Distinctive reflectance patterns occur where regions of differing reflectances meet, namely at boundary segments and vertices (Figure 6.3). In the following description surface patches defined by vertices are used, but the same principles apply to surface patches defined by line segments or curves. Selecting landmarks from the model, then, corresponds to searching the locale network for vertices (equivalently lines or curves) which are surrounded by regions which form a distinctive pattern. The structure of the network makes this computation relatively straightforward and efficient. The locale containing the agent is known because the location of the agent is expressed in terms of that locale and its coordinate system.

agent-location = (locale pose)

This locale, together with its offspring (freespace and the objects contained in the locale), determine patterns which may be seen by the agent. The agent's view will be a subset of the patterns present on the inside surface of the locale and the outside surfaces of its offspring. The search for landmarks is limited

to the network description of these surfaces. The procedure for selecting landmarks from the model is as follows:

1. From the locale corresponding to the agent's current location collect all the vertices associated with the regions on the inside surface of the locale and the outside surface of each offspring locale. This is accomplished by following the pointers in the surface description for that locale (Figure 6.4):
2. Delete vertices which are not expected to be visible to the agent from its current location. This is done by first clipping the projection to the image plane (ignoring occlusion) and then deleting occluded vertices from what remains.
3. Discard vertices which are not part of the common boundary of at least two regions which differ in reflectivity by some threshold. These reflectivities can be found by following pointers to the associated regions (Figure 6.5)
4. Return the remaining vertices.

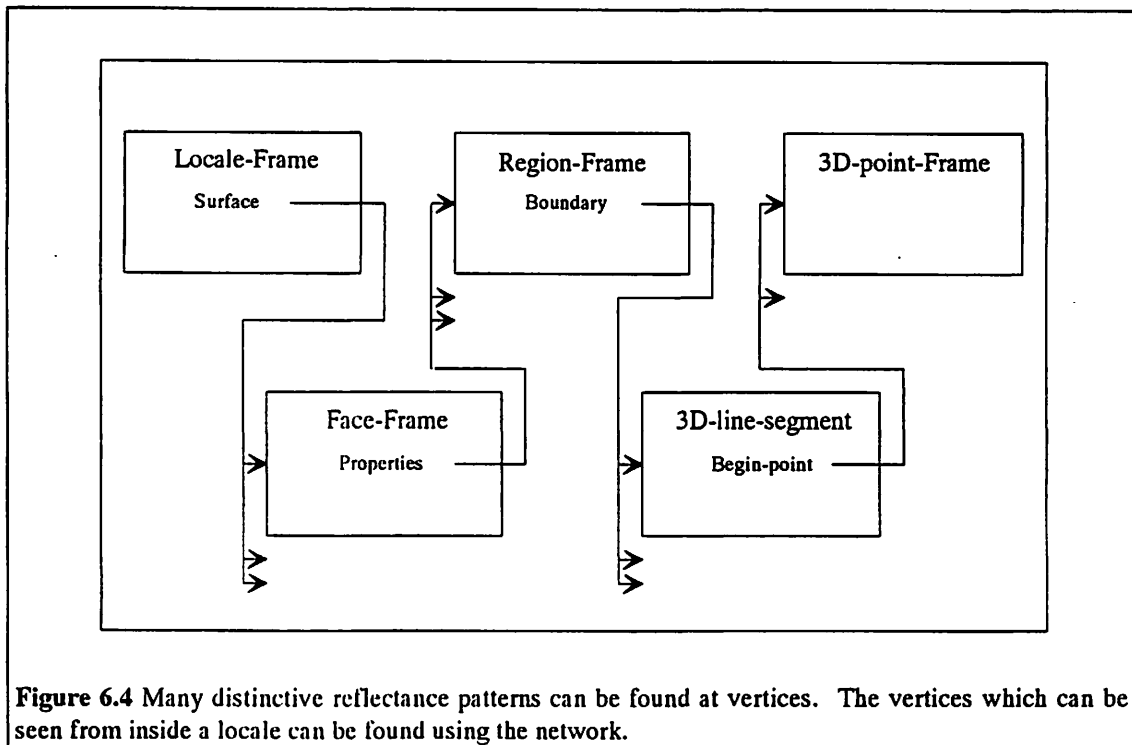


Figure 6.4 Many distinctive reflectance patterns can be found at vertices. The vertices which can be seen from inside a locale can be found using the network.

Each vertex returned by this procedure identifies a surface patch defined by the regions which have that vertex on their boundary. These are the landmarks used both in action-level and plan-level perceptual servoing. In both types of servoing, knowledge of the reflectances of these landmarks is used to construct the appearance templates which are then matched to the image data to identify the landmarks in the image.

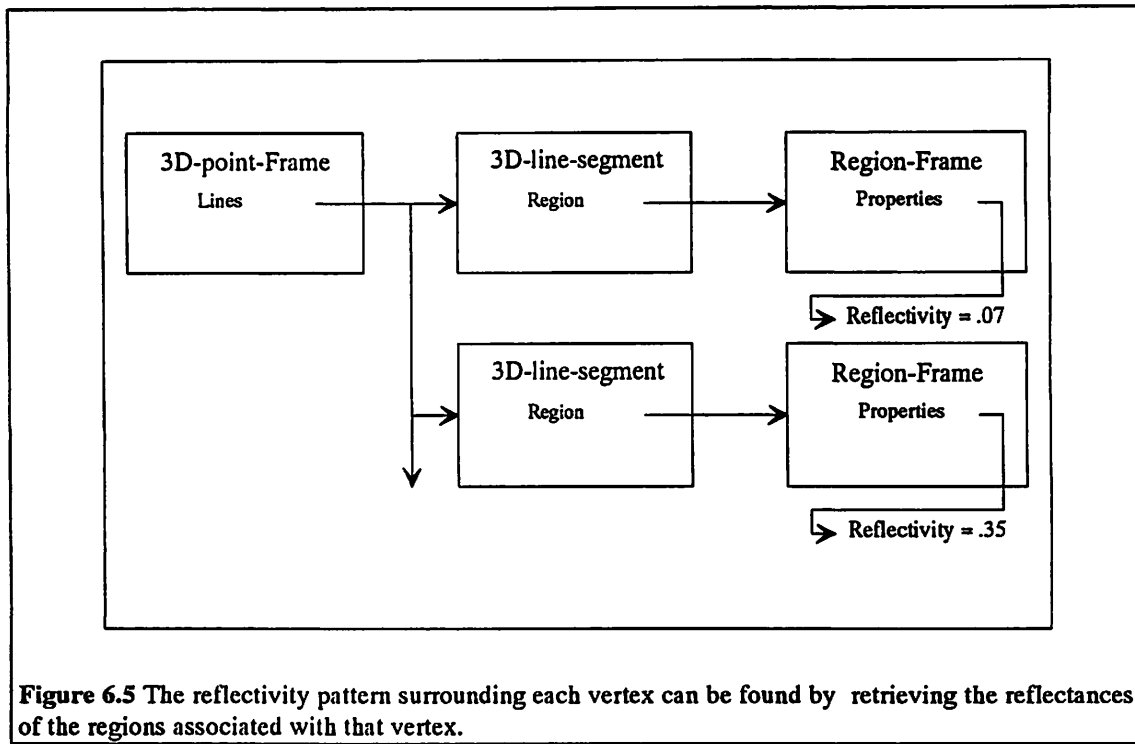


Figure 6.5 The reflectivity pattern surrounding each vertex can be found by retrieving the reflectances of the regions associated with that vertex.

6.2.2 Constructing Appearance Templates from the Model

Appearance templates are image arrays which specify how a landmark should appear in the image. For the vertex landmarks selected by the procedure described in the previous section, these are $n \times n$ image arrays centered on the image of the vertex with pixel values determined by the geometry, reflectance values and the agent's location as represented in the model. Constructing these arrays is a localized rendering process.

Templates are constructed by ray tracing, considering only those regions whose boundaries pass through the vertex. For each landmark vertex this is accomplished as follows:

1. Collect the regions which have the vertex on their boundary. This is accomplished by following pointers in the locale network, beginning with the 3D-point frame associated with the vertex, as illustrated in Figure 6.5.
2. For each pixel in the appearance template there is a ray which starts at the camera lens center and passes through the center of the pixel. Find the 3D intersection of this ray with each of the regions collected in (1).

If the ray does not intersect any of the regions either the patterns associated with the landmark are too detailed and may be subject to aliasing or the vertex is on an occluding edge of an object and

certain pixels will be unpredictable. The use of the landmark will be error prone so it should not be used. In this case a null template is returned.

Otherwise assign to the pixel the value $255 \cdot R$, where R is the reflectance of the region whose point of intersection with the ray is closest to the camera lens focal point.

3. The resulting array is the appearance template.

The resulting template is used match the landmarks with their projections in an image. This matching is done using correlation in a way which matches reflectance values, rather than intensities, to reduce the ever present effects of uneven illumination.

6.2.3 Matching Appearance Templates to the Data Using Correlation

Correlation is a well understood mathematical tool which has been widely used in signal processing and in 2D image processing, but it has not been used as much in 3D scene processing because there are several severe problems which arise. It is easy to describe these problems if we think of an image as a function $f(x,y)$ on the x - y plane. Correlation is a measure of how similar two such functions are. Images of 3D scenes are, however, strongly effected by several factors, all of which are significant in the kinds of scenes our agent will encounter.

1. The shape of the image of an object varies as the viewpoint is changed, due to projection effects.
2. Changes in viewpoint alter what can be seen in the image of a scene, due to occlusion effects.
3. Changes in lighting modify the intensity of the image, either locally or globally.
4. Specularities vary, sometimes strongly, with lighting and viewpoint

All four problems affect the nature of the image function $f(x,y)$ corresponding to a scene in such a way that its "shape" and "height" may vary considerably. This makes correlation useless for global scene matching and makes local scene matching difficult, at best. On the other hand, knowledge of the agent's approximate location, together with knowledge of how these factors affect the image function makes it possible to cope with these problems. Problems 1 and 2 are managed by the way the appearance templates are constructed. The perspective distortions of 1 and the occlusion effects of 2 are kept to a minimum by the ray tracing procedure and the rejection test in step 2 of the appearance template construction method described in Section 6.2.2.

The third problem is managed by transforming each acquired image $P(i,j)$ in such a way that the transformed image $RP(i,j)$ is independent of the scene illumination. This new image is the result of dividing each pixel by the sum of the pixels in a window surrounding it.

$$RP(i,j) = \frac{P(i, j)}{\sum_{k,l \in W} P(i+k, j+l)}$$

Under the assumption that for each sufficiently small patch of surface in the environment intensity is constant. This is not an unreasonable assumption for an environment which is not highly textured by shadows. The values of each pixel $P(i,j)$ in such a patch can be expressed as:

$$P(i,j) = I * \rho(i,j) \quad 0 \leq i, j \leq n$$

where I is the (constant) light intensity over the surface patch and $\rho(i,j)$ is the average reflectance of the surface area which images at $P(i,j)$. Under these circumstances the effects of the illumination can be removed by dividing the value of each pixel in the image by the sum of the pixel values in a small patch surrounding that pixel. The new reflectance pixel array $RP(i,j)$ has values given by the formula:

$$\begin{aligned} RP(i,j) &= \frac{P(i, j)}{\sum_{k,l \in W} P(i+k, j+l)} = \frac{I * \rho(i, j)}{\sum_{k,l \in W} I * \rho(i+k, j+l)} \\ &= \frac{I * \rho(i, j)}{I * \sum_{k,l \in W} \rho(i+k, j+l)} = \frac{\rho(i, j)}{\sum_{k,l \in W} \rho(i+k, j+l)} \end{aligned}$$

where k,l range over the set $W = \{k,l \mid -w_1 \leq k \leq w_1 \text{ and } -w_2 \leq l \leq w_2\}$ for some w_1 and w_2 . Clearly, the pixel values in this new array are independent of the illumination incident on the surface. They are determined by the reflectance properties of the surface.

Management of the fourth problem, specularities, has been left as a subject future research. In the experimental system it was assumed that the number of landmarks unrecognized or improperly recognized due to specularities would be small. This assumption is not always justified; but, although specularities exist in the experimental environment used in this thesis, they did not cause problems during the experiments.

Matching the appearance templates to image data is done using normalized correlation. First the appearance template $T(i,j)$ and the acquired image $P(k,l)$ are transformed to the intensity independent images $RT(i,j)$ and $RP(k,l)$. Then the transformed template $RT(i,j)$ is matched against the transformed image $RP(k,l)$ using what is known as normalized correlation. The value of this

$$NC(i,j) = \frac{2 * \sum_{k,l \in W} RP(k+i,l+j) * RT(k+w_1, j+w_2)}{\sum_{k,l \in W} RP(k+i,l+j)^2 + \sum_{k,l \in W} RT(k+w_1, j+w_2)^2}$$

The computation of this value can be made more efficient by rearranging the terms in the expression.

$$\begin{aligned} & 2 * \sum_{m,n \in W} \left[\frac{P(m+i, n+j)}{\sum_{k,l \in W} P(k+i, l+j)} \right] * \left[\frac{T(m+w_1, n+w_2)}{\sum_{k,l \in W} T(k+w_1, l+w_2)} \right] \\ = & \frac{\sum_{m,n \in W} \left[\frac{P(m+i, n+j)}{\sum_{k,l \in W} P(k+i, l+j)} \right]^2 + \sum_{k,l \in W} \left[\frac{T(m+w_1, n+w_2)}{\sum_{k,l \in W} T(k+w_1, l+w_2)} \right]^2}{\sum_{m,n \in W} \left[\frac{P(m+i, n+j)}{\sum_{k,l \in W} P(k+i, l+j)} \right]^2 + \sum_{m,n \in W} \left[\frac{T(k+w_1, l+w_2)}{\sum_{k,l \in W} T(k+w_1, l+w_2)} \right]^2} \\ = & \frac{2 * \left[\frac{1}{\sum_{k,l \in W} P(k+i, l+j)} \right] * \left[\frac{1}{\sum_{k,l \in W} T(k+w_1, l+w_2)} \right] * \sum_{m,n \in W} P(m+i, n+j) * T(m+w_1, n+w_2)}{\sum_{m,n \in W} \left[\frac{P(m+i, n+j)}{\sum_{k,l \in W} P(k+i, l+j)} \right]^2 + \sum_{m,n \in W} \left[\frac{T(k+w_1, l+w_2)}{\sum_{k,l \in W} T(k+w_1, l+w_2)} \right]^2} \\ = & \frac{2 * \sum_{m,n \in W} P(m+i, n+j) * T(m+w_1, n+w_2)}{R1 * \sum_{m,n \in W} P(m+i, n+j)^2 + R2 * \sum_{m,n \in W} T(k+w_1, l+w_2)^2} \end{aligned}$$

where R1 and R2 are the ratios:

$$R1 = \left[\frac{\sum_{k,l \in W} T(k+w_1, l+w_2)}{\sum_{k,l \in W} P(k+i, l+j)} \right] \text{ and } R2 = \left[\frac{\sum_{k,l \in W} P(k+i, l+j)}{\sum_{k,l \in W} T(k+w_1, l+w_2)} \right] = \frac{1}{R1}$$

These ratios need only be computed once for each value of (i,j) , resulting in a more efficient computation. This correlation method has proven to be very reliable for locating landmark images in the indoor environments used for the experiments described below.

Once the selected landmarks have been located in the image, the image data has been matched to the 3D landmark models. Next appropriate corrective actions are taken. These actions are different for action-level and plan-level servoing, so we describe them separately.

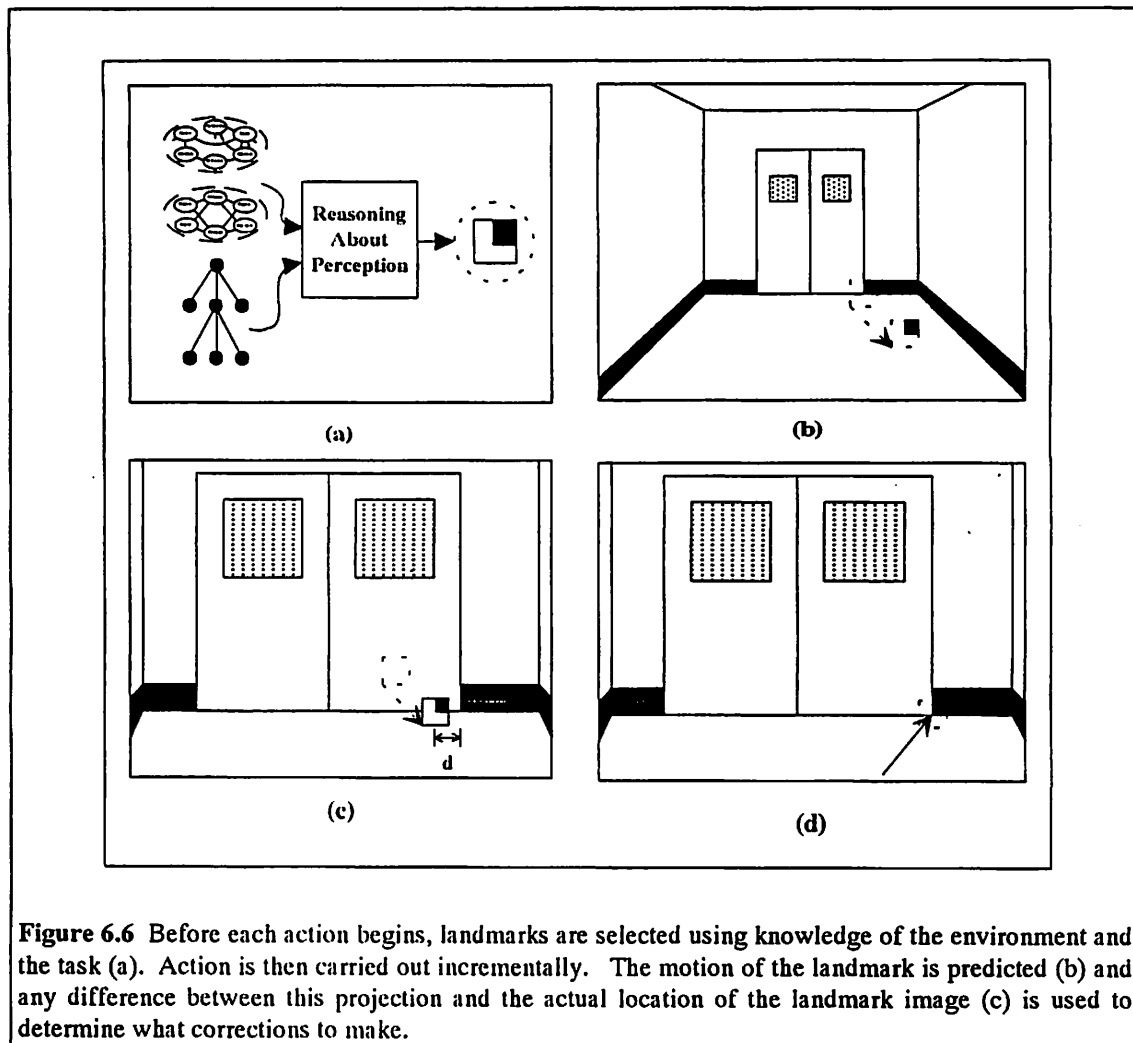


Figure 6.6 Before each action begins, landmarks are selected using knowledge of the environment and the task (a). Action is then carried out incrementally. The motion of the landmark is predicted (b) and any difference between this projection and the actual location of the landmark image (c) is used to determine what corrections to make.

6.2.4 Action-Level Perceptual Servoing

Primitive actions are carried out incrementally, using the location of landmark images to compute necessary corrections. Each increment begins by selecting landmarks and matching their projections

with data in the image. By measuring the discrepancy between where the features should be and where they are determined to be in the image, it is possible to compute the corrective action required to bring the positions into agreement (Figure 6.6). This perceptual servoing has the effect of locking the robot onto a trajectory which improves the accuracy of the actions over that which would be obtained without servoing.

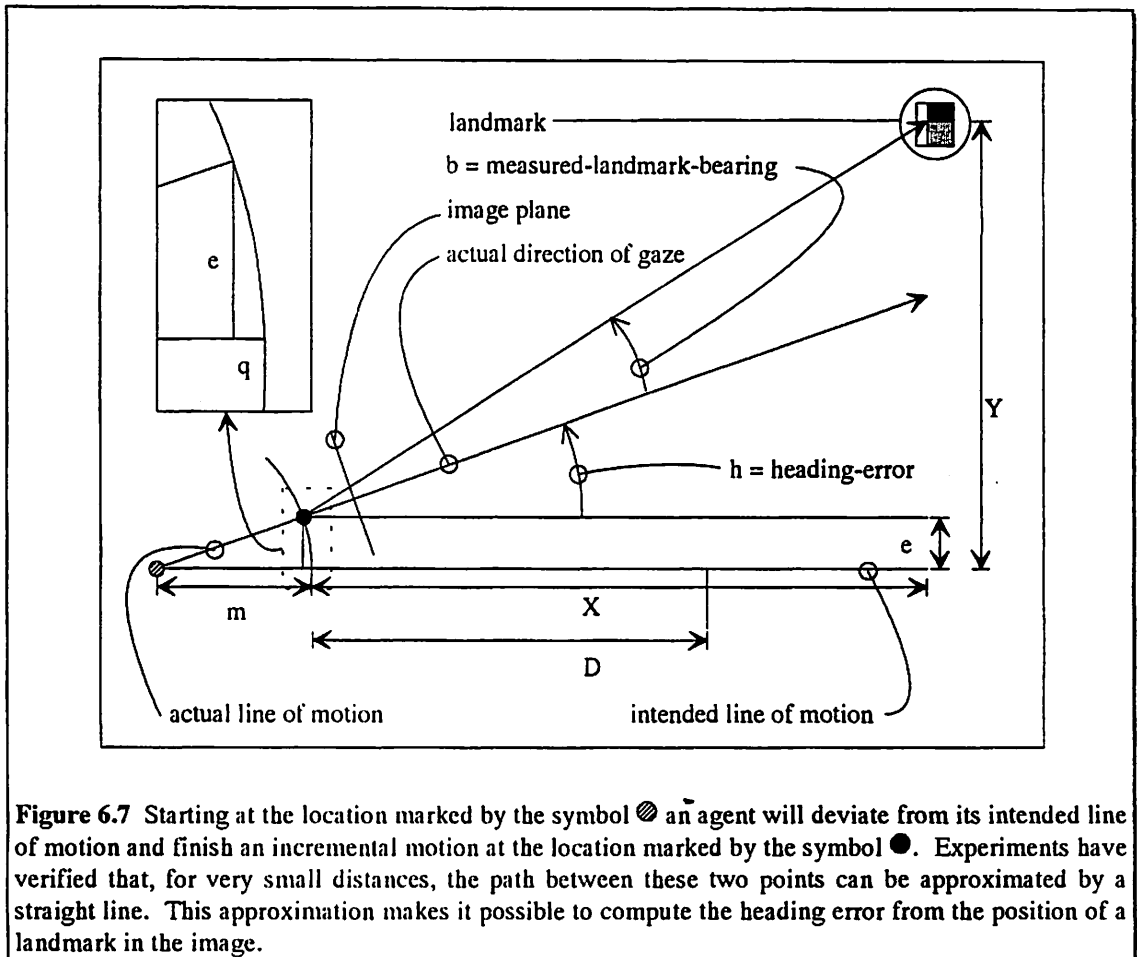


Figure 6.7 Starting at the location marked by the symbol \odot an agent will deviate from its intended line of motion and finish an incremental motion at the location marked by the symbol \bullet . Experiments have verified that, for very small distances, the path between these two points can be approximated by a straight line. This approximation makes it possible to compute the heading error from the position of a landmark in the image.

The "move" action illustrates the action-level perceptual servoing principle. Experiments have shown that during an open loop execution of an intended straight line forward motion the robot vehicle moves in a curved path. Sufficiently small incremental paths can be approximated by a straight line at some angle with respect to the intended line. Figure 6.7 depicts the geometry of the situation.

If we let

m = incremental distance moved
 e = distance "off desired line" after incremental move of m
 q = shortfall in distance covered along intended line.
 x = x-coordinate of the landmark image (image coordinates)
 f = focal length of camera lens
 X = x-coordinate of landmark in robot coordinates (expected)
 Y = y-coordinate of landmark in robot coordinates (expected)
 h = heading error after the incremental move.
 b = measured landmark bearing

Then

$$h + b = \tan^{-1} \left(\frac{Y-e}{X+q} \right)$$

As a consequence, since $\tan(b) = (-x/f)$, we see that:

$$h = \tan^{-1} \left(\frac{Y-e}{X+q} \right) + \tan^{-1} \left(\frac{-x}{f} \right)$$

Thus from the observed x-coordinate of the landmark image we can estimate the heading error, provided the values of e and q are small enough so that

$$\left(\frac{Y-e}{X+q} \right) \approx \left(\frac{Y}{X} \right)$$

This is typically the case for small incremental motions. In our configuration e is on the order of .02 ft and q is on the order of .05 ft. Hence this approximation is reasonable for landmarks more distant than 3 feet.

Given an approximation to the heading error the corrective action is determined according to the geometry indicated in Figure 6.8. The quantities e and q are given by

$$e = m \cdot \sin(h)$$

$$q = e \cdot \tan(h)$$

The agent's motion is corrected by steering to a point on the intended line of travel a distance D away from the location predicted for this incremental motion. The correction can be computed, using e and q as follows:

$$\text{correction} = - \left[h + \tan^{-1} \left(\frac{e}{D+q} \right) \right]$$

$$= - \left[h + \tan^{-1} \left(\frac{m \sin(h)}{D + m \sin(h) \tan(h)} \right) \right]$$

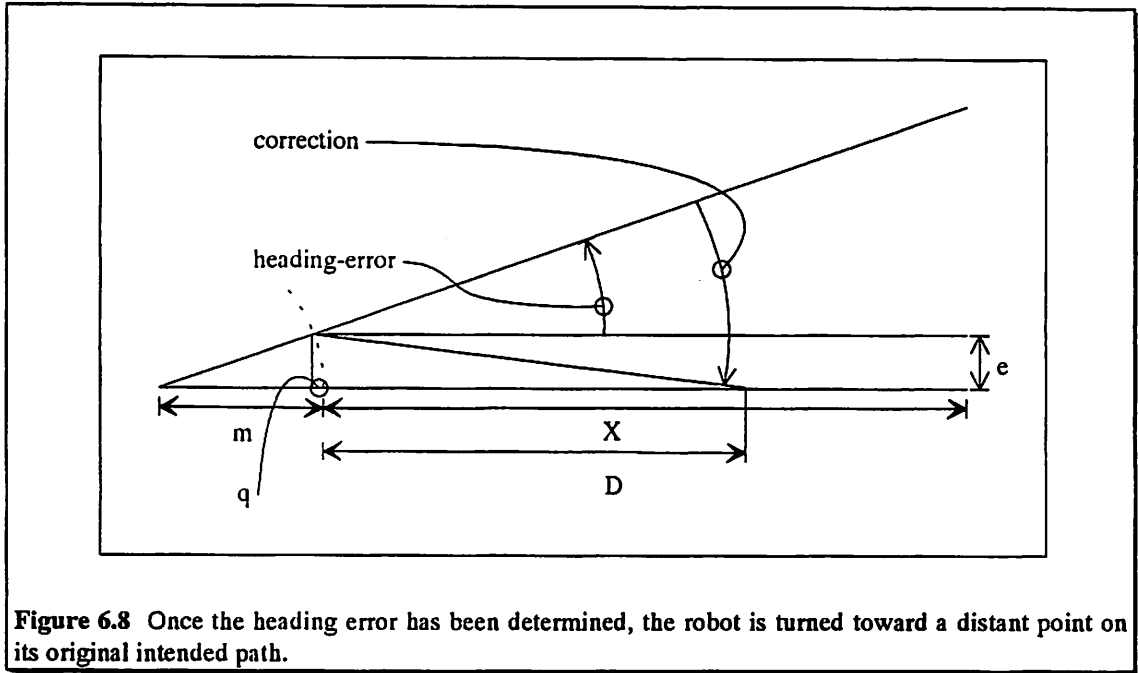


Figure 6.8 Once the heading error has been determined, the robot is turned toward a distant point on its original intended path.

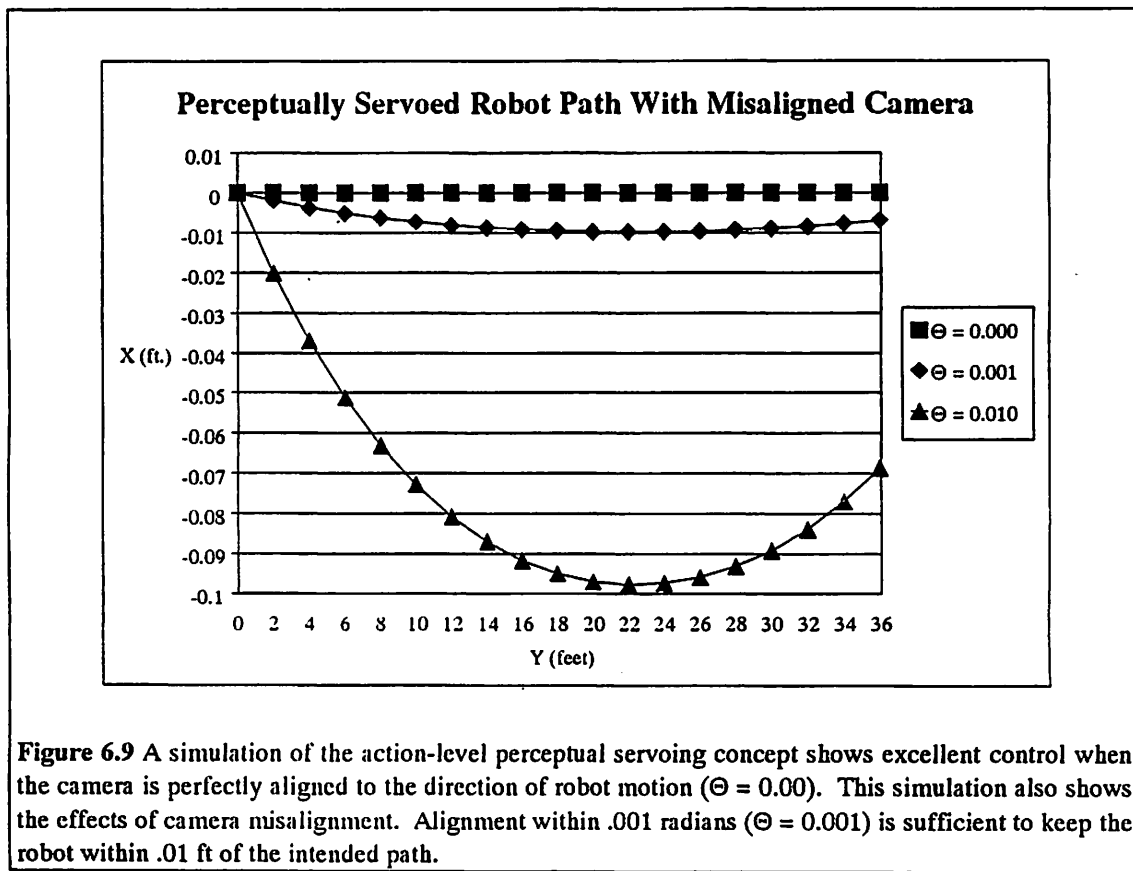
In this analysis it has been assumed that the z-axis of the camera is perfectly aligned with the robot's direction of motion. In practice this alignment is very difficult to achieve so a simulation was performed to determine the effect of camera misalignment on the accuracy of the robot's trajectory.

The effect of a camera misalignment angle of Θ has the effect of giving an erroneous measurement of the heading error h . Taking the misalignment into account the expression for h should be:

$$h = \tan^{-1} \left(\frac{Y-e}{X+q} \right) + \tan^{-1} \left(\frac{-X}{f} \right) + \Theta.$$

In the simulation a hypothetical robot was moved using the action-level perceptual servoing method described above. When the heading error was computed it was assumed that $\Theta = 0$ and this value was used to correct the agent's estimated heading H . The simulated motions assumed a heading of $(H - \Theta)$. Figure 6.9 shows simulated trajectories for alignment errors $\Theta = 0.000$, $\Theta = 0.001$ and $\Theta = 0.010$ radians. These results predict a trajectory error of less than 0.1 feet (1.2 inches) provided Q is kept below 0.01 radians (.57 degrees). This indicates that errors due to misalignment can be controlled.

Several experiments, both with and without action-level perceptual servoing, have been run. In these experiments Harvey was to move 40 feet in increments of 2 feet along a straight line marked on the floor of a Graduate Research Center hallway. After every incremental move the deviation from the marked line was measured. This experiment was run a number of times; the results shown in the graph of Figure 6.10 represent the best unservoed result compared with a typical servoed result. Even after a rather painstaking setup procedure the unservoed vehicle wandered over two inches from the line within the first 20 feet. Other unservoed trials resulted in as much as a foot deviation in the 20 foot distance. Unservoed trials were stopped at around 20 feet because the vehicle was significantly off course and the total deviation was increasing. In contrast, in servoing mode the vehicle stayed within .3 inch of the line over its entire length of 40 feet.

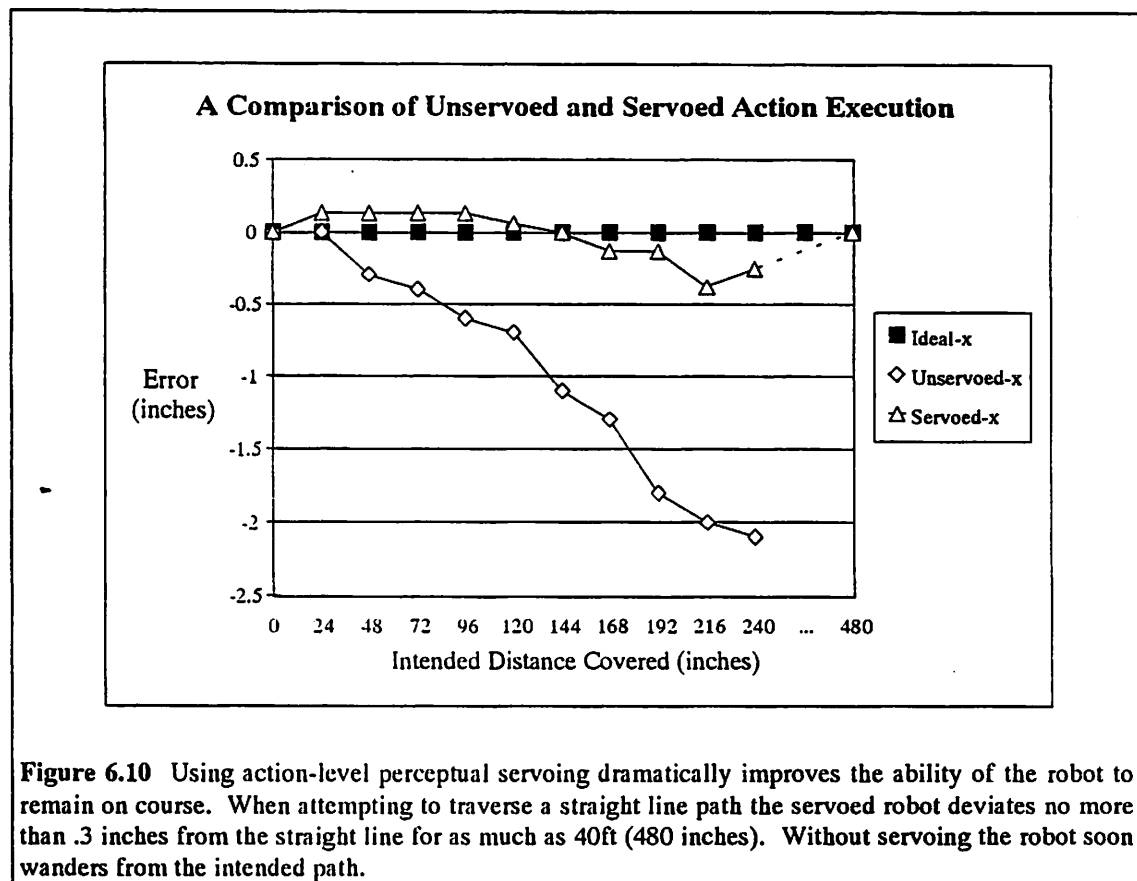


These experiments indicate that unservoed motion can result in significant error during the execution of a single primitive subgoal. On the other hand, perceptual servoing can provide the accuracy necessary to

control motion over reasonably large navigation segments. On the basis of these experiments, it seems reasonable to assume that each primitive action can be carried out accurately enough to offer significant control over the amount of replanning necessary. They make a strong case for the validity of the assumption put forth in support of the arguments made in section 5.2.3.

6.2.5 Plan-Level-Perceptual Servoing

Plan-level perceptual servoing uses the same landmark selection and matching procedures as action level perceptual servoing. The resulting matches and the 3D model information are used to determine the robot's pose, using a 3D pose determination algorithm [Kumar and Hanson, 1989, 1990]. If the robot location, as determined by the pose computation agrees with what is expected then the milestone has been satisfied and there is no need to modify any subgoals.



If pose refinement and expectations differ by a small amount, then the first location in the next subgoal is adjusted to reflect this difference. The next subgoal is changed from

(ptrans location-2 location-3)

to

(ptrans pose-determined-location location-3).

Plan-and-monitor automatically performs a detailed plan refinement if this change makes it necessary. In the event that the pose-determined location differs greatly from what was expected, or if it should fail to determine a location, control is passed to goal-level perceptual servoing.

	y0-6	y0-3	y0	y0+3
Expected				
x	40.00	40.00	40.00	40.00
y	3.50	3.75	4.00	4.25
z	0.00	0.00	0.00	0.00
theta-x	0.00E+00	0.00E+00	0.00E+00	0.00E+00
theta-y	0.00E+00	0.00E+00	0.00E+00	0.00E+00
theta-z	3.14E+00	3.14E+00	3.14E+00	3.14E+00
Measured				
x	39.93	39.93	40.11	39.85
y	4.01	4.01	4.08	4.17
z	-0.22	-0.22	0.20	0.00
theta-x	-4.69E-03	-4.69E-03	-9.00E-03	-4.95E-03
theta-y	-1.51E-02	-1.51E-02	-5.00E-03	-9.25E-03
theta-z	3.14E+00	3.14E+00	3.14E+00	3.14E+00

Figure 6.11 Using correlation for landmark matching, 3D pose determination experiments have shown the ability to determine the robot's location to within 1.5 inches. These results compare the expected (incorrect) location with what was measured. The ground truth location in all cases was the same $x=40$ feet, $y=4$ feet, $z=0$ feet, $\theta_x=0$, $\theta_y=0$ and $\theta_z=3.1415$ radians.

To use position information in this way, how accurate must the pose returned by the pose determination step be? Since the robot has been servoing on image features over the previous navigation leg, it is likely that it will be fairly close to its expected position, say within 6-12 inches. Within this area

of uncertainty, pose refinement should return a more accurate value, say within an inch or two. Outside of this area of uncertainty, it is likely that significant subgoal redevelopment will be necessary, so it is appropriate for plan level servoing to pass control up to goal-level servoing.

Experimental results support this accuracy requirement. Figure 6.11 shows the results from an experiment to examine the accuracy of this approach. Using a single image taken in a hallway, the expected location of the robot was varied to create the effect of being off target. The figure shows the measured pose for different expected locations, corresponding to the actual location of the robot, indicated in the column marked y_0 , and three incorrect expected locations, respectively labeled: y_0-3 meaning the agent thinks it is 3 inches to the left; y_0-6 , meaning 6 inches to the left and y_0+3 ; meaning 3 inches to the right.

The system as a whole has been successfully run along several short paths in the Research Center. On the most complex path, one from a point inside the robot laboratory, through a doorway, to a point in the hallway just outside Allen's office, the vehicle stopped within 1 inch of the final goal. The elapsed time for this experiment was about 10 minutes.

6.3 Remarks About Complexity

As mentioned in section 6.2, plan-level and action-level servoing are very similar; both have the following structure:

1. Select Landmarks (Section 6.2.1)
2. Construct Templates (Section 6.2.2)
3. Match Templates to Data (Section 6.2.3)
4. Use this information:
 - a) to correct heading while performing action-level servoing (Section 6.2.4) or
 - b) to adjust the plan sketch plan-level servoing (Section 6.2.5)

The first three of these steps are identical for the two servoing operations. The difference between action-level and plan-level servoing lies in what is done with the results of the matching operation.

6.3.1 The complexity of landmark selection, template construction and matching

Landmark selection complexity is determined by the complexity of the locale as measured by the number of faces m and vertices n in that locale. This can be easily seen by reviewing the steps outlined in section 6.2.1.

Step

Complexity

1. Collect all the vertices from the locale. Let n be the number of these vertices. This can be done in constant time if the vertices are explicitly kept on a list.

$O(c)$

2. Delete vertices which cannot project onto the agent's "retina". This can be accomplished by determining the location of the projection of each selected vertex in the agent's image plane. The complexity of the projection and clipping operation is constant and must be performed no more than n times, once for each vertex selected in step 1.

$O(n)$

3. Delete vertices which are occluded from view. This process is done by determining whether or not any of the faces of the locale or its offspring intersect a line from the lens center to the vertex. If we let m be the total number of faces of the locale and all its offspring, then each vertex remaining after step 2 will be compared with no more than m faces.

$O(n+m)$

4. Delete vertices which are not perceptually distinctive. Distinctiveness is determined by analyzing the regions which intersect at the vertex. For each vertex remaining after step 3 the difference between the maximum and minimum reflectance of the regions surrounding it. Since the regions associated with a vertex can be found in constant time using the features of the locale network, this operation is of complexity $O(n*t)$ where t is the maximum number of faces which surround the vertex. In principle t is unbounded. In practice, however, it is rare that a vertex is common to more than four faces. It is reasonable to assume that $t < 5$.

$O(n)$

Hence the landmark selection complexity is:

$O(n+m)$.

Template construction is done for each of the n' vertices selected as landmarks. Since there are no more than n visible vertices, $n' \leq n$. The complexity of template construction is therefore:

For each selected landmark vertex do :

Complexity

1. Collect the regions associated with that vertex. This can be done in bounded time for each vertex by following the pointers in the network.

$O(n')$

2. Compute each of the $k \times l$ values for the template by ray tracing using reflectance values obtained from the regions surrounding the vertex. If there are t faces associated with the vertex, the complexity of this operation is $O(k*l*t)$. If the template size is kept constant or is bounded (in the experiments they were always smaller than 10×10) then the complexity for each vertex is $O(t)$. Finally, due to the finite resolution of the template, vertices with a very large number of regions associated with them will no

longer be distinctive, since much of their features will be blurred. As a result, the computation required for this computation is bounded.

$$O(c)$$

The complexity of template construction for all landmark vertices is therefore:

$$O(n')$$

Matching templates to image data:

For each landmark do the following:

Complexity

1. For each landmark vertex compute the correlation of the template with the image for each pixel position in a $p \times q$ window and return the coordinates of the pixel position corresponding to the highest correlation value. The template is $k \times l$ pixels, so this is of complexity $O(k \cdot l \cdot p \cdot q)$. Again assuming the template size is bounded we see that matching complexity is .

$$O(n' \cdot p \cdot q)$$

The complexity of matching is:

$$O(n' \cdot p \cdot q)$$

The complexity of landmark selection and template construction are both determined by the local complexity of the locale as measured by the number of faces m and vertices n . Using these component operations we can now determine the complexity of the two servoing operations.

6.3.2 The complexity of action-level perceptual servoing

The objective of action-level perceptual servoing is to insure that each primitive action is executed accurately. This is accomplished by detecting errors as soon as possible and making corrections to prevent the errors from growing. The complexity of an iteration of the action-level servoing loop can be determined as follows:

<u>Step</u>	<u>Complexity</u>
Select landmarks	$O(n \cdot m)$
Take a picture	$O(c)$
For each landmark	
Construct a template	$O(n') = O(n)$
Match each template in the image	$O(n' \cdot p \cdot q) = O(n \cdot p \cdot q)$
Compute heading error indicated by each landmark, average the results and make the correction	$O(n') = O(n)$
	<hr style="width: 50%; margin-left: auto; margin-right: 0;"/>
Thus the complexity is	$O(\max(n \cdot m, n \cdot p \cdot q))$.

In the light of the analysis of Section 6.2.4, each incremental action should be small. If we let s be the number of such increments per foot, then for large s the motion increment between corrections will be small enough that agent can be modeled as traveling along a circular arc. This means that for sufficiently large s , the values of p and q need not be larger than:

$$p = a*(1/s)$$

$$q = b*(1/s)$$

where a and b are experimentally determined constants. Thus the complexity of an iteration of the loop can be expressed as:

$$O\left[\max\left(n*m, n * \frac{a}{s} * \frac{b}{s}\right)\right] = O\left[\max\left(n*m, \frac{n*a*b}{s^2}\right)\right] = O\left[\max\left(n*m, \frac{n}{s^2}\right)\right]$$

This expression indicates that, as s increases, the cost of the loop is determined by n*m, which is an expression of the complexity of the environment in terms of the number of vertices and faces. As implemented, this environment is the locale which contains the path of the primitive action, so the cost of the loop is determined by the complexity of the *local* environment.

The successful performance of action-level servoing process *depends* on the fact that it is a fine grained RML process. The simplicity of the "reasoning" and "looking" steps in this process have been *made possible* by the fact that they interact in a fine grained manner. Both the reasoning and looking in this loop are simple because it is known that nothing will change very much.

6.3.3 Complexity of plan-level perceptual servoing

The complexity of plan level perceptual servoing is simplified by the frequent use of perception during the action-level servo loop. It can be determined by analyzing the algorithm:

<u>Step</u>	<u>Complexity</u>
Select Landmarks	O(n*m)
Take a picture	O(c)
For each landmark	
Construct a template	O(n') = O(n)
Match the template to image data	O(n'*p*q) =
O(n*p*q)	
Determine the pose from the matched data.	O(n') = O(n) ¹
Adjust the plan sketch by replacing the start location for the next subgoal	O(c)
Refine the resulting subgoal, if it is not a primitive subgoal	O(log(ρ)) ²

Because action-level and plan-level servoing uses the same landmark selection technique . This means that the p x q search window for each milestone can be very small, since the uncertainty in the location

¹ The algorithm used to determine pose in this system, pose-points [Kumar and Hanson, 1989, 1990] has a complexity of O(n'*i) where n' is the number of landmarks used to determine the pose and "i" is the number of iterations required by the algorithm to settle on a solution. Experiments indicate that i<5.

² This result was developed in Chapter 5.

of each landmark in the image is the same as for action-level matching. Using the same arguments as in section 6.3.2 the complexity of the loop, excluding goal refinement, can be expressed as:

$$O\left(\max\left[n*m, \frac{n*a*b}{s^2}\right]\right) = O\left(\max\left[n*m, \frac{n}{s^2}\right]\right)$$

As for the action-level, the cost for this portion of the loop is determined by the local complexity of the environment.

The arguments in Chapter 5 showed that planning complexity was reduced by the fine grained RML architecture. These arguments were predicated on the assumption that monitoring actions with perception would improve their accuracy. The experiments in Section 6.2 on action-level servoing support this assumption.

The complexity of "moving" in RML is simplified by the fine grained architecture. Since each move corresponds only to a single primitive subgoal, rather than a multiple goal path, the reasoning done by action-level servoing is greatly simplified. The path specified by a primitive subgoal will be entirely contained within a small number of locales, usually one. This places bounds on m and n and consequently on the expression $O\left(\max\left[s*n*m, \frac{n}{s}\right]\right)$, the complexity of attaining a primitive subgoal.

Finally, milestone recognition is also simplified by the fine grained architecture. The arguments of Section 6.3.3 show that frequent monitoring of primitive actions reduces the complexity of milestone recognition, the "look-a-little" step of the plan-level servoing loop. Thus reasoning (planning and predicting) action and looking (matching) are all simplified by the fine grained RML architecture.

6.3.4 Goal-Level Perceptual Servoing

Although the goal-level perceptual servoing loop was not implemented as a part of this work, a rough sketch of how it might be accomplished gives some insight as to how RML could affect its complexity.

This servo loop can be summarized by the following algorithm:

1. Construct a plan sketch.
2. Invoke plan-level servoing.
3. If the goal has been achieved quit
otherwise go to 4 (an error has occurred or the agent is lost)
4. Determine the agent's current location and go to 1.

Plan sketching and plan-level servoing have already been discussed. The completion of this loop would be accomplished by implementing the module identified in Chapter 1 as the "Where-am-I?" module. The objective of this module is to determine the current location of the agent. There are two parts to this task:

1. Determine which locale contains the agent. The agent is always located in the "free space" sublocale of some locale. It may be in the free space of the hallway, or the robot laboratory, or some other locale. The first task is to determine this locale.
2. Determine the pose of the agent in the coordinate system of the locale identified in step 1.

The locale and pose determined by these steps specify the location frame required to formulate the goal required to begin the replanning process of the goal level servoing loop described in Section 3.2.3.3.

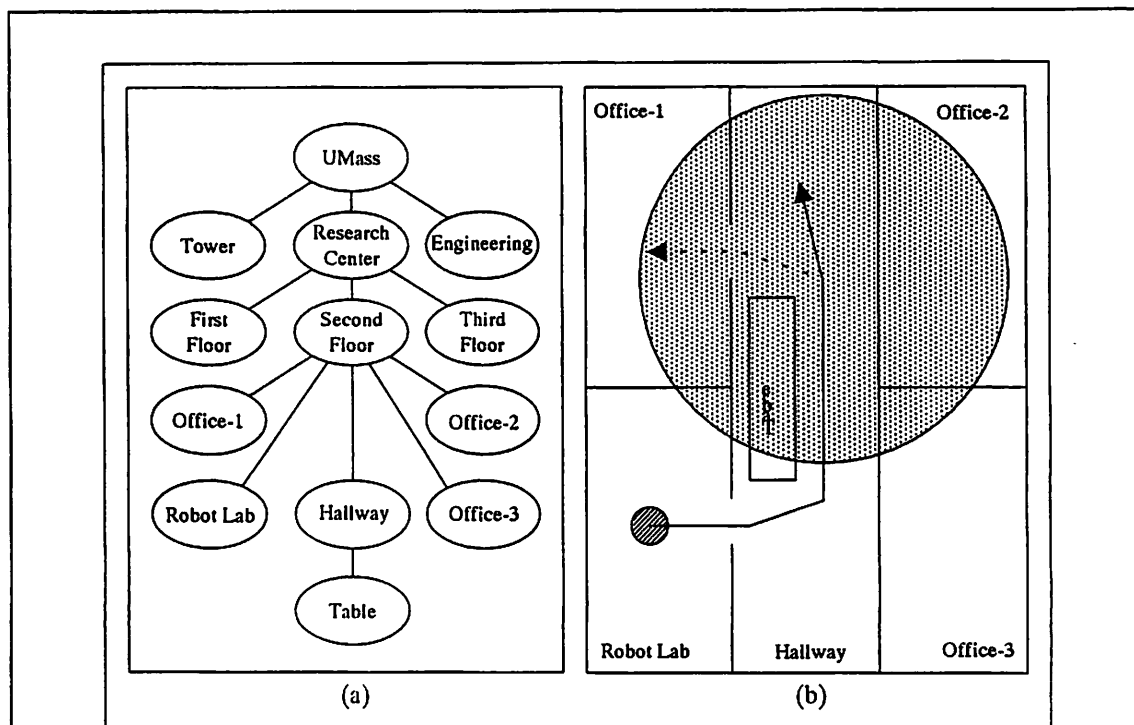


Figure 6.12 The "Where am I?" function required for goal-level perceptual servoing is seen as using the contained-by hierarchy (a) to guide a search of the network for the locale whose free space sublocale contains the agent. This process begins at the root node of the hierarchy and works its way down the tree, reasoning about the properties of sublocales to devise tests which use the sensors to eliminate alternatives. The fine grained nature of the RML architecture makes it possible to limit the scope of this search. As described in the text, a circle of uncertainty (b) can be used to define a scope mechanism which is similar to the one used in sketching plans (Chapter 5). The least common locale of the locales which intersect this circle can be used as a scope-locale. For the situation in (b) the scope locale would be the Second Floor locale shown in (a).

Identifying the locale is a process which uses the contained-by hierarchy of the network to identify the appropriate locale by systematically eliminating alternatives. The details of this procedure will require the development of new reasoning and sensing techniques, but an example will illustrate the approach.

Beginning at the root of the contained-by hierarchy, identified as UMass in Figure 6.12 (a), the first step is to determine whether the agent is in the free space of the UMass locale or one of its offspring. By reasoning about the properties of these four locales it may be determined that a quick elimination test would be:

```
if (the floor of the locale is green)
  then locale = UMass
  else if (the walls of the locale are not beige)
    then locale = engineering
    else locale = research center.
```

If the results of these tests indicate that the locale is the research center then the tower locale, the engineering locale and their descendants have been eliminated. This process would continue recursively to determine the floor locale and the room locale. Once the room locale is identified the process will turn to looking for major features of the locale, such as a window or door, to determine its rough position and orientation. Finally, a more accurate pose would be determined using a procedure like the one described in Section 6.2.5.

Designing this procedure will require the development of many new techniques. Doing so has been left as a task for future research. The description given in the previous paragraph, however, presents enough detail to indicate that reasoning about the environment will benefit from focusing tools like the scope mechanism described in Chapters 3 and 5. The RML mechanism as described in the preceding chapters makes this possible.

The net effect of the action-level and plan-level servoing mechanisms is to keep knowledge about the agent's location current and low in uncertainty. The "Where-am-I?" procedure is invoked as soon as plan-level servoing fails. This is detected at the completion of a *primitive* subgoal and, by design, the agent's location was known before this subgoal was attempted. This information can be used to specify a circle (sphere) of uncertainty. The last known location would serve as the center of this circle and the

length of the primitive subgoal would be used as a basis for determining its radius (Figure 6.12 (b)). It is very likely that the agent's location is in one of the locales intersected by this circle (sphere). A scope mechanism for the "Where-am-I?" procedure can be defined as choosing the least common ancestor of these locales as a scope-locale. It is conjectured that effect of using this mechanism in the locale search will result a complexity reduction as dramatic as that achieved by the scope mechanism used in planning.

CHAPTER 7

SUMMARY AND FUTURE RESEARCH

As was discussed in Chapter 1, the work described in this dissertation has had two goals: to make some contributions to basic research in Artificial Intelligence (AI) and to contribute to the development of a theory for AI applications. The principal contribution to AI research has been to offer a better understanding of the interaction between reason, action and perception in AI systems. The work and contributions in this area are summarized in Section 7.1. The experimental system developed during this investigation shows real time potential and is made up of a number of techniques which have potential use in application environments. These possibilities are summarized in Section 7.2

7.1 Artificial Intelligence Research

As was shown in Chapter 2, the conventional view of an intelligent system has been that it is decomposed into a set of macromodules, including those associated with perception and reasoning. This view has resulted in a coarse grained interaction between reasoning, perception and action. The position taken in Chapter 1 was that a tighter interaction between these activities would be beneficial. Specifically, the claim was made that:

Intelligent behavior can be demonstrated and efficiently executed using traditional AI techniques by introducing a *fine grained* integration or "interweaving" of these three activities. By performing these activities incrementally, doing only what is necessary to begin the next process, the system as a whole becomes more efficient because questions are posed and answered in a timely fashion and because the questions are more focused, making it easier for the next process to do its job.

Chapter 3 then described a general view of how such a system would work and presented an overview of the specific system used to explore the idea. The details of this system, the RML system, along with a description of the results, were presented in Chapters 4,5 and 6. The results offer strong support for the central claim and the system implementation details make additional contributions in representation, planning and vision.

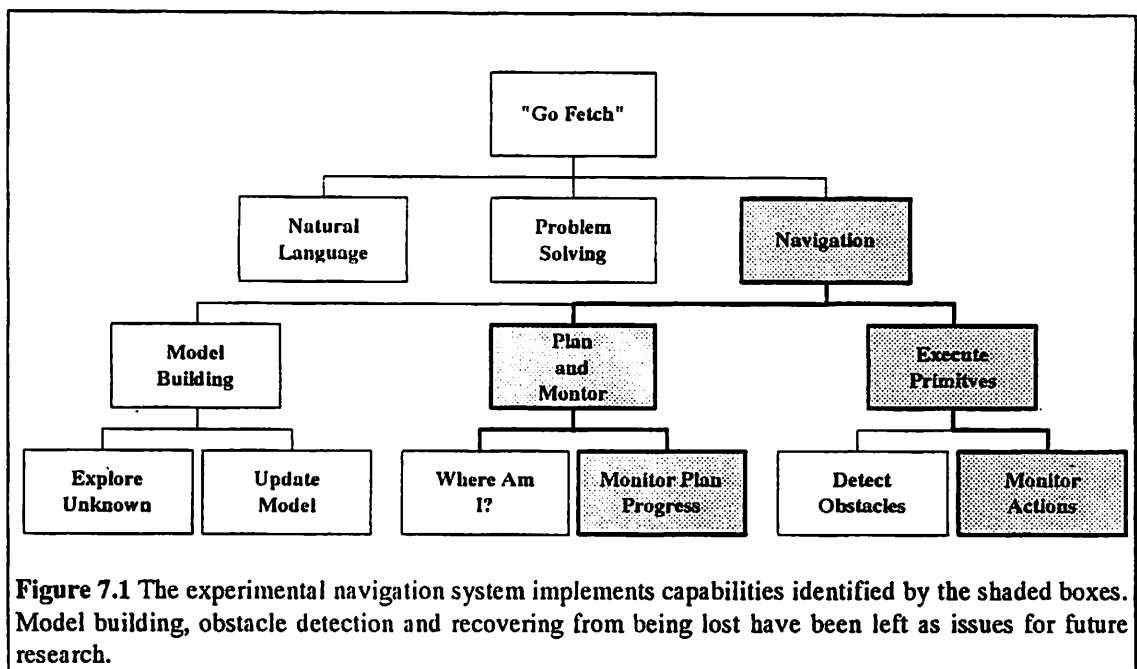
Chapter 4 described a frame-based representation of space and used it to introduce the concept of a "contained-by" hierarchy. This contained-by hierarchy made it possible to define mechanisms, referred to as "scope", "perspective" and "location". The scope and perspective mechanisms can be used to limit

the volume of space and the amount of detail considered by reasoning processes. The location mechanism is used when describing position at various levels of abstraction. Shapes in this network are described in terms of planar faces. Extending the network to include curved surfaces and generic spaces has been left for future research.

Planning, as described in Chapters 3 and 6, was done incrementally and made use of the scope and perspective mechanisms. The use of scope and perspective resulted in a geometrically defined hierarchical planning algorithm which produces plans in $O(\rho)$ operations (ρ = number of path segments in the plan). This represents a dramatic reduction in planning effort when compared to the complexity of an algorithm like A* and realizes the theoretical prediction that is possible for a hierarchical planner to have linear complexity [Korf, 1987]. It was argued in Chapter 5 that, even when this new algorithm is employed, execution errors and unplanned events would cause replanning often enough to result in a total planning complexity of $O(\rho^2)$ if a macroprocess architecture is used. An analysis of the fine-grained RML structure showed that the incremental nature of its plan development reduced this complexity to $O(\rho \log(\rho))$ and the interweaving of planning and perception further reduced this to $O(\rho)$.

Chapter 3 introduced an approach to perception which uses sensors in a way which is analogous to the way a blind man uses his cane. This approach, together with the fine-grained RML architecture, made it possible to use simple vision techniques to control the action of the robot in two nested servoing loops called "action-level" and "plan-level" perceptual servoing. The results in Chapter 6 demonstrated that action-level perceptual servoing was capable of maintaining a straight line path to within 1/3 inch for a distance 40 feet and that plan-level servoing was capable of controlling the uncertainty of the robot's position to a few inches. Given an environment modeled by n vertices and m faces, each invocation of the vision process was accomplished in $O\left[\max\left[n \cdot m, \frac{n}{s^2}\right]\right]$ operations (where s = the number of times this process was repeated per foot). In addition to making this type of vision possible, interweaving reason and perception plays a role in placing bounds on this quantity. Interweaving means that vision is invoked frequently, so s tends to be large. Moreover, frequent reasoning keeps the information about location current and thus makes it possible to keep m and n small.

These results give strong evidence in support of the central claim in the navigation application. The complexity of planning and vision were both reduced as a result of interweaving reason, action and perception. As was mentioned in Chapter 1, however, methods for obstacle detection using vision, determining the location of a lost agent (Where am I?), and object recognition were not implemented in this study, nor was there any attention given to the exploration of unknown environments (see Figure 7.1). It is left as a matter of future research to extend this work to perform these functions and analyze the complexity of the resulting system.



7.2 Applications Research

The goal of making a contribution to the development of AI applications in industry was one of the primary factors which shaped the specific design of the experimental system. The locale network was designed as a feasible extension to models currently used in Computer Aided Design (CAD). The overall system was aimed to ultimately be used in applications like the one illustrated in Figure 7.2. In applications such as this, the model of the environment (parts and the workstation) is accurate and is produced by a CAD design team. Consequently, the existence and precision of the model used in this study are not at issue. The challenge presented by the application-oriented goal of this project has been

to discover methods which can take advantage of such a model to accomplish industrial tasks, subject to the real time, environmental and economic constraints faced in that arena.

The techniques described in Chapters 3 - 6 have met that challenge in the navigation application. It was shown how the RML system could use such a model to accomplish planning and vision tasks in reasonable time, using reasonable computing equipment. In addition, both vision and planning were done in such a way that only the model need be changed to solve this problem for a new navigation environment. It is believed that these planning and vision techniques can be adapted to assembly and inspection as well. Doing so has been left as a topic for research.

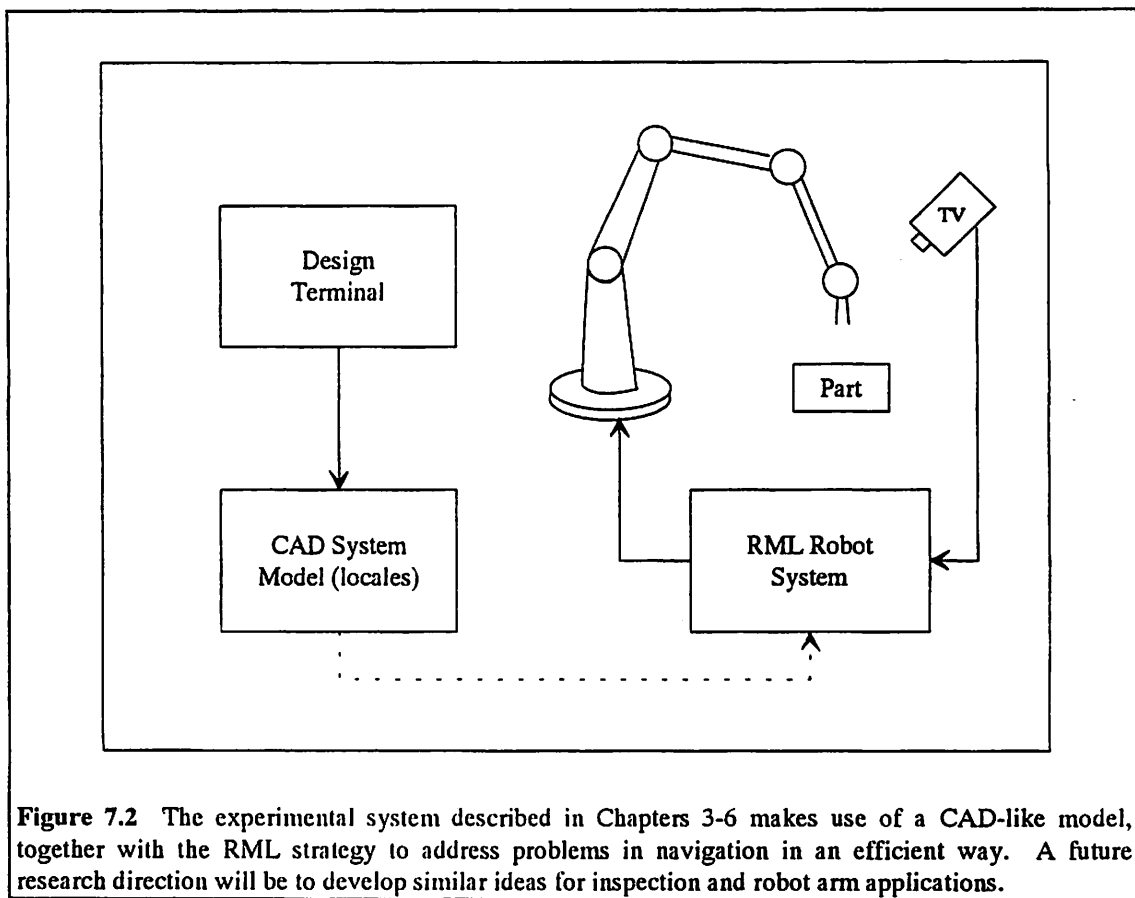


Figure 7.2 The experimental system described in Chapters 3-6 makes use of a CAD-like model, together with the RML strategy to address problems in navigation in an efficient way. A future research direction will be to develop similar ideas for inspection and robot arm applications.

BIBLIOGRAPHY

- Amerinex Artificial Intelligence Inc. "KB-Vision Users Guide". Amerinex Artificial Intelligence Inc, Amherst, MA, 1989.
- Amerinex Artificial Intelligence Inc. "KB-Vision Programmers Reference". Amerinex Artificial Intelligence Inc, Amherst, MA, 1989.
- Arkin, Ronald C. "Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments." Phd. Dissertation, University of Massachusetts, September 1987.
- Baird, M.L. "SIGHT-1: A computer vision system for automated chip manufacture. IEEE Transactions on Systems, Man and Cybernetics, SMC-8(2): 133-139, 1978
- Ballard, D.H. and Brown, C.M. "Computer Vision." Englewood Cliffs, New Jersey: Prentice-Hall Inc, 1982.
- Ballard, D.H., Brown, C.M. and Feldman, J.A. "An approach to knowledge directed image analysis". In A.R. Hanson and E. M. Riseman (Eds.), Computer Vision Systems, pp 271-281, New York: Academic Press, 1978.
- Barrow, Harry G. and Tenenbaum, J. Martin. "Recovering Intrinsic Scene Characteristics from Images". Artificial Intelligence Center Technical Note 157, SRI International, 1982.
- Beveridge, J. R., Weiss, R. and Riseman, E. M. "Optimization of 2-Dimensional Model Matching Under Rotation, Translation and Scale." COINS Department, University of Massachusetts (Amherst), Technical Report No. 89-57, June, 1989.
- Beveridge, J.R., Weiss, R. and Riseman E.M. "Combinatorial Optimization Applied to Variable Scale 2D Model Matching." 10th International Conference on Pattern Recognition, Atlantic City, NJ, June 1990.
- Brice, C.R. and Fennema C.L. "Scene Analysis Using Regions". Artificial Intelligence 1(3), pp205-226, Fall 1970.
- Brooks, Frederick P., "The Mythical Man-Month", Addison Wesley, 1978
- Brooks, R.A. "Symbolic Reasoning Among 3-D Models and 2-D Images". Phd. dissertation, Stanford University, 1981.
- Brooks, R.A. "Model based three dimensional interpretations of two dimensional images". IJCAI 7, pp. 619-624, 1981.
- Brooks, Rodney A. "A Robust Layered Control System For a Mobile Robot". IEEE Journal of Robotics and Automation, Vol RA-2, No. 1., March 1986.
- Brooks, Rodney A. "Intelligence Without Representation". MIT Research Report, MIT Artificial Intelligence Laboratory, Cambridge Mass., 1987.
- Brooks, Rodney A. "Planning is just a way of avoiding figuring out what to do next". MIT Working Paper 303, MIT Artificial Intelligence Laboratory, 1987.

Chapman, David and Agre, Philip E. "Abstract reasoning as emergent from concrete activity". In Michael P. Georgeff and Amy L. Larsky (Eds.), "Reasoning About Actions and Plans", Los Altos, CA: Morgan-Kaufman, 1987.

Choudhary, A. N. "A reconfigurable and Hierarchical Parallel Processing Architecture: Performance Results for Stereo Vision". Proceedings of the 10th International Conference on Pattern Recognition, Volume II, pp 389-393, IEEE Computer Society Press, Los Alamitos, CA: 1990.

Control Data Corporation. "INTERSYS 103 Users Guide". Minneapolis: Control Data Corporation, Professional Services, 1983

Control Data Corporation. "XAMIN Users Manual". Minneapolis: Control Data Corporation, Professional Services, 1984.

Dickmans, E. "Visual Dynamic Scene Understanding Exploiting High-Level Spatio-Temporal Models". Proceedings of the 10th International Conference on Pattern Recognition, Volume II, pp 373-378, IEEE Computer Society Press, Los Alamitos, CA: 1990.

Dickmans, E. and Graefe, V. "Applications of Dynamic Monocular Machine Vision". Machine Vision and Applications, Vol. 1, pp. 241 and following, 1988a.

Dickmans, E. and Graefe, V. "Dynamic Monocular Machine Vision," Machine Vision and Applications, Vol. 1, pp. 223-240, 1988b.

Draper, Bruce A., Collins, Robert T., Brolio, John, Hanson, Allen R. and Riseman, Edward. "The Schema System." International Journal of Computer Vision, pp 209-250, Volume 2, No. 3, January 1989.

Duda, R. O. "Some Current Techniques for Scene Analysis". Technical Note 20, Artificial Intelligence Center, SRI International, Menlo Park California, 1970.

Ernst, G., and Newell, A. "GPS a case study in generality and problem solving". New York: Academic Press, 1969.

Fennema C., Riseman E. and Hanson, A. R. "Planning With Perceptual Milestones to Control Uncertainty in Robot Navigation." Proceedings of SPIE -- International Society for Photographic and Industrial Engineering, Mobile Robots III, Cambridge, MA, pp 2-18, 1988.

Fennema, C.L., Hanson, A.R. and Riseman, E. "Towards Autonomous Mobile Robot Navigation," COINS Department, University of Massachusetts (Amherst), Technical Report No. 89-104, October 1989. Also appeared in the Proceedings of the DARPA Image Understanding Workshop, pp 219-231, May 1989

Fennema, C.L., Hanson, A.R., Riseman, E.M., Beveridge, J.R. and Kumar, R. "Towards Autonomous Navigation." IEEE Transactions on Systems, Man and Cybernetics, Vol. 20, No. 6., 1990.

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

Fikes, R.E., Hart, P.E., and Nilsson, N.J. "Learning and executing generalized robot plans". Artificial Intelligence Journal 2, 1971.

Fukushima, T. "A Survey of Image Processing LSIs in Japan". Proceedings of the 10th International Conference on Pattern Recognition, Volume 2, pp 394-401, IEEE Computer Society Press, Los Alamitos, CA: 1990.

Garvey, Thomas D. "Perceptual Strategies for Purposive Vision". SRI Technical Note 117, 1976.

Gleason, G. J. and Agin, G. J. "A modular vision system for sensor-controlled manipulation and inspection". Proceedings of the 9th International Symposium and Exposition on Industrial Robots, pp. 57-70, Washington D. C., March 1979.

Green, C. "Application of theorem proving to problem solving. Proceedings of the first International Joint Conference on Artificial Intelligence, pp 291-239, 1969. Also appears in Readings in Artificial Intelligence, Webber, Bonnie Lynn and Nils Nilsson (Eds.), pp. 202-222, Palo Alto, CA: Tioga Publishing Company, 1981.

Guzman, A. "Decomposition of a Visual Scene into Three-Dimensional Bodies". Proceedings of the FJCC, pp 291-304, December 1968.

Hanson, Allen R. and Riseman, Edward M. "Visions: A Computer System for Interpreting Scenes". In Computer Vision Systems, Allen R. Hanson and Edward M. Riseman (eds.), Academic Press, 1978.

Hanson, Allen R. and Riseman, Edward M. "The VISIONS Image Understanding System". In C. Brown (ed.), Advances in Computer Vision, Erlbaum Associates, 1987.

Hart, P.E., Nilsson, N.J. and Raphael, B. "A formal basis for the heuristic determination of minimum cost paths". IEEE transactions on Systems Science and Cybernetics, SSC-4(2), 100-107, 1968.

Hart, P.E., Nilsson, N.J., and Raphael, B. Correction to "A formal basis for the heuristic determination of minimum cost paths". SIGART Newsletter, No. 37, pp. 28-29, December 1972.

Hendler, James, Tate, Austin and Drummond, Mark. "AI Planning: Systems and Techniques". AI Magazine, pp. 61-77, Summer 1990.

Herman, Martin and Albus, James S. "Real-time Hierarchical Planning for Multiple Mobile Robots". Proc. DARPA Knowledge-Based Planning Workshop, Austin, Texas, pp. 22-1 - 22-10, December 1987.

Herman, Martin and Albus, James S. "Overview of the Multiple Autonomous Underwater Vehicles (MAUV) Project". Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia, PA., pp 618-620, April 1988.

Herman, Martin and Albus, James S. "Overview of MAUV: Multiple Autonomous Undersea Vehicles". Unmanned Systems, Vol. 7, No. 1, pp 36-52, Winter 1988-89.

Herman, M., Hong, T., Swetz, S., Oskard, D. and Rosol, M. "Planning and World Modeling for Autonomous Undersea Vehicles." Proceedings of the Third IEEE International Symposium on Intelligent Control, Arlington, VA, August 1988.

Herman, Martin, Albus, James S. and Hong, Tsai-Hong. "Intelligent Control for Multiple Autonomous Undersea Vehicles." In Neural Networks for Control, W.T. Miller and R. Sutton, eds, Cambridge, MA: MIT Press, 1990.

Herman, M., Kanade, T. and Kuroe, S. "Incremental Acquisition of a Three-Dimensional Scene Model from Images." S IEEE PAMI, pp. 331-340, 1984.

Holland, S.W., Rossol, L., and Ward M.R. "CONSIGHT-I: A vision-controlled robot system for transferring parts from belt conveyors". In G.G Dodd and L. Rossol (Eds.), pp. 81-100, Computer vision and sensor based robots. New York: Plenum, 1979.

- Kanade, T. "Model representations and control structures in image understanding". *IJCAI 5*, pp1074-1082, 1977.
- Kleinrock, L. and Kamoun, F. "Hierarchical routing for large networks". *Comp. Networks 1*, pp. 155-174, 1977.
- Korf, R. E. "Planning as Search: A Quantitative Approach." *Artificial Intelligence 33*:65-68, 1987.
- Kosslyn, Stephen M. "Image and Mind". Cambridge, Massachusetts: Harvard University Press, 1980.
- Kuan, Darwin, Phipps, Gary and Hsueh, A.-Chaun. "Autonomous Robotic Vehicle Road Following". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 10, No 5., September 1988.
- Kuipers, B. "Modeling Spatial Knowledge". *Cognitive Science 2*:129-153, 1978.
- Kuipers, Benjamin. "The 'Map in the Head' Metaphor". *Environment and Behavior*, Vol 14, No. 2, pp. 202-220, March 1982.
- Kuipers, Benjamin J. and Byun, Yung-Tai, "A Robust, Qualitative Approach to a Spatial Learning Mobile Robot". *Proceedings of the SPIE Cambridge Symposium on Advances in Intelligent Robotics Systems*, November 1988a.
- Kuipers, Benjamin J. and Byun, Yung-Tai. "A Robust, Qualitative Method for Robot Spatial Learning". *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI-88)*, 1988b.
- Kumar, R. and Hanson, A. "Robust Estimation of Camera Location and Orientation from Noisy Data having Outliers". *Proceedings of the IEEE Workshop on Interpretation of 3D Scenes, Austin Texas*, pp. 52-60, November 1989. A more detailed version may be found in Department of Computer and Information Sciences, TR89-120, University of Massachusetts, Amherst: December 1989.
- Kumar, R. and Hanson, A. "Pose Refinement: Application to Model Extension and Sensitivity to Camera Parameters." Department of Computer and Information Sciences, University of Massachusetts, TR90-112, Amherst: December 1990.
- Lawton, D.T., Levitt, T.S., McConnel, C.C, Nelson, P.C. and Glicksman, J. "Environmental Modeling and Recognition for an Autonomous Land Vehicle". *Proceedings of the DARPA Image Understanding Workshop*, 1987.
- Lewis, Harry R. and Papadimitriou, Christos H. "Elements of the Theory of Computation". Englewood Cliffs, New Jersey: Prentice-Hall, 1981.
- Lowrie, J. Thomas, M., Gremban, K., and Turk, M. "The Autonomous Land Vehicle (ALV) Preliminary Road-Following Demonstration". *Proceedings of Intelligent Robots and Computer Vision (SPIE 579)*, D.P. Casasent, Ed., pp. 336-350, 1985.
- Lozanno-Perez, Thomas and Wesley, Michael A. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles." *Communications of the ACM*, Volume 22, Number 10, pp 560-570, October 1979.
- Marr, David. "Vision". San Francisco: Freeman, 1982.
- Matsuyama, T. "Knowledge-based aerial image understanding systems and expert systems for image processing". *IEEE Transactions on Geoscience and Remote Sensing 25*, pp. 305-316, 1987.

- Matsuyama, T. and Hwang, V. "SIGMA: a framework for image understanding - integration of bottom-up and top-down analyses". IJCAI Proceedings, pp. 908-915, 1985.
- McCulloch, W. S. and Pitts, W. "A logical calculus of the ideas immanent in nervous activity." Bulletin of Mathematical Biophysics, 5, pp 115-133, 1943.
- McKeown, D.M. Jr. "Building knowledge-based systems for detecting man-made structures from remotely sensed imagery". Philosophical Transactions of the Royal Society of London A 324, pp. 423-435, 1988.
- McKeown, D.M., Harvey Jr., W.A. and McDermott, J. "Rule-based interpretation of aerial imagery". T- PAMI 7, pp. 570-585, 1985.
- Miller and Johnson-Laird. "Language and Perception". Cambridge Mass: Belknap Press of Harvard University Press, 1976
- Minsky, Marvin and Pappert, Seymour. "Perceptrons: An Introduction to Computational Geometry". Cambridge: MIT Press, 1969.
- Minsky, Marvin. "The society of mind". New York: Simon and Schuster, 1986.
- Moravec, Hans P. "The Stanford Cart and the CMU Rover". IEEE Proceedings, Vol 71, No. 7, July 1983.
- Mulder, J.A., Mackworth, A.K. and Havens, W.S. "Knowledge structuring and constraint satisfaction: the Mapee approach". T-PAMI 10, pp. 866-879, 1988.
- Nagao, M., Matsuyama, T. and Ikeda, Y. "Region extraction and shape analysis of photographs". IJCP 4, pp. 620-628, 1978.
- Nagao, M., Matsuyama, T. and Ikeda, Y. "Structural analysis of complex aerial photographs". IJCAI 4, pp 610-616, 1979.
- Nilsson, Nils J. "Principles of Artificial Intelligence". Morgan Kaufmann, Los Altos, California, 1980.
- Nilsson, Nils J. "Shakey the Robot". SRI Technical Note 323, 1984.
- Newell, Allen and Simon, Herbert. "Human Problem Solving". Englewood Cliffs, New Jersey: Prentise Hall Inc, 1972.
- Newell, A., Shaw, J. C., and Simon, H. A. "Empirical explorations with the logic theory machine: A case history in heuristics." In Computers and Thought, E. A. Feigenbaum and J. Feldman (Eds.), pp 109-133, New York: McGraw Hill, 1963.
- Newell, A. and Simon, H. A. "GPS, a program that simulates human thought". In Computers and Thought, E. A. Feigenbaum and J. Feldman (Eds.), pp 279-293, New York: McGraw Hill, 1963.
- Ohta, Y. "A region oriented image-analysis system by computer". Doctoral dissertation, Information Science Department, Kyoto University, 1980.
- Oskard, D.N., Hong, T.-H and Shaffer, C.A. "A Spatial Mapping System for Autonomous Underwater Vehicles". Proceedings of the SPIE Conference on Sensor Fusion: Spatial Reasoning and Scene Interpretation, pp 439-450, Cambridge, MA, November 1988.

- Pappert, Seymour. "One AI or Many?". *Daedalus, Journal of the American Academe of Arts and Sciences*, pp 1-14, special issue on Artificial Intelligence, Vol. 117, No. 1, Winter 1988.
- Pomerleau, Dean A. "Neural Network Based Autonomous Navigation". Appeared in "Vision and Navigation", Charles Thorpe (ed.), Kluwer Academic Publishers, 1990.
- Quam, L. "The ImageCalc vision system". Stanford Research Institute, Menlo Park, California, 1984.
- Raphael, Bertram. "The Thinking Computer: Mind Inside Matter." W. H. Freeman and Company, New York, 1976.
- Riley, K., McConnel, C. and Lawton, D.T. "Powervision Users Manual". Advanced Decision Systems, Mountain View, California. 1988.
- Roberts, L. "Machine Perception of three dimensional solids". In J. Tippett (Ed.) *Optical and Electro-Optical Information Processing*, Cambridge Mass: MIT Press, 1965.
- Rosenblatt, F. "Principles of neurodynamics". New York: Spartan, 1962.
- Rubin, S. "The ARGOS image understanding system". Doctoral dissertation, Computer Science Department, Carnegie Mellon University, 1978.
- Rummelhart, David E. and McClelland, James L. "Parallel Distributed Processing." Cambridge MA: MIT Press, 1988.
- Sacerdoti, Earl D. "Planning in a hierarchy of abstraction spaces". *Artificial Intelligence* 5, pp 115-135, 1974.
- Sacerdoti, Earl D. "A structure for plans and behavior". Technical Note 109, Artificial Intelligence Center, SRI International, Inc., Menlo Park, California, 1975.
- Schank, Roger and Abelson, Robert. "Scripts Plans Goals and Understanding". Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1976.
- Selfridge, O.G. "Pattern recognition in modern computers." *Proceedings of the Western Joint Computer Conference*, 1955.
- Shari, Y. "Recognition of real world objects using edge cues". In A. R. Hanson and E. M. Riseman (Eds.), *Computer Vision Systems*, pp 353-362, New York: Academic Press, 1978
- Shepard, Roger N. and Cooper, Lynn A. "Mental Images and their Transformations". Cambridge, Massachusetts: MIT Press, 1982.
- Thorpe, C., S. Shafer, and A. Stentz. "An Architecture for Sensor Fusion In A Mobile Robot". *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol 3, pp. 1622 and following, April 1986.
- Thorpe, Charles, Herbert, Martial H., Kanade, Takeo and Shafer, Steven A. "Vision and Navigation for the Carnegie-Mellon NAVLAB". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 10, No 3, May 1988.
- Toscani, G. and Faugeras, O. "Structure from Motion Using the Reconstruction and Reprojection Technique". *Proceedings of the IEEE Workshop on Computer Vision*, pp. 345-348, 1987.

Tsotsos, John K. "Knowledge organization and its role in representation and interpretation for time-varying data." *Computing Intelligence* 1, pp. 16-32, 1985.

Turk, Matthew A., Morgenthaler, David G., Gremban, Keith D. and Marra, Martin. "VITS - A Vision System for Autonomous Land Vehicle Navigation". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 3, May 1988.

Waxman, Allen M., LeMoigne, Jacqueline J. LeMoigne and Davis, Larry S. et al. "A Visual Navigation System for Autonomous Land Vehicles". *IEEE Journal of Robotics and Automation*, Vol RA-3, No 2, April 1987.

"Webster's New Collegiate Dictionary." C. & G. Merriam Company, Springfield, Massachusetts, 1977.

Weems, C. C., Rana, D., Hanson, A. R., and Riseman, E. M. "An Overview of Architecture Research for Image Understanding at the University of Massachusetts". *Proceedings of the 10th International Conference on Pattern Recognition*, Volume 2, pp 379-384, IEEE Computer Society Press, Los Alamitos, CA: 1990.

Weiss, Lee E., Sanderson, Arthur C. and Neuman, Charles P. "Dynamic Sensor-Based Control of Robots with Visual Feedback". *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 5, October 1987.

Weymouth, T. "Using Object Descriptions in a Schema Network for Machine Vision". Ph.D. Dissertation, COINS Technical Report 86-24. Department of Computer and Information Science, University of Massachusetts, Amherst, Ma., May 1986.

Williams, Thomas D. "Image understanding tools". *Proceedings of the International Conference on Pattern Recognition*, pp. 606-610, Los Alamitos, CA: IEEE Computer Society Press, June 1990.

Zhang, Z. and Faugeras, O. "Building a 3D World Model with a Mobile Robot: 3D Line Segment Representation and Integration". *Proceedings of the 10th International Conference on Pattern Recognition*, Volume I, pp 38-42, IEEE Computer Society Press, Los Alamitos, CA: 1990.