Interweaving Reason, Action and Perception[1]

Claude L. Fennema, Jr.
Computer Science Program
Mount Holyoke College, South Hadley, MA 01075
and
Allen R. Hanson
Computer and Information Science Department
University of Massachusetts, Amherst MA, 01003

Abstract

In an attempt to understand and emulate intelligent behavior Artificial Intelligence researchers have historically 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 paper describes an investigation into the effect of reducing this architecture granularity. It shows 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 paper introduces the "reason a little, move a little, look a little" paradigm (RML), describes an RML implementation, presents analytical arguments that show the RML scheme can result in significant reduction in complexity for both planning and vision, and describes experiments that indicate that the RML concept can work in practice. The complexity of each invocation of vision in the RML system is shown to be $O(\max(n*m, 1/(s^2)))$ where $n*m$ is a measure of the complexity of the *local* environment and s is the number of vision invocations per foot. It is also shown 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 $O(\rho)$. The experiments presented show that the RML system can perform in a variety of environments, realizes the efficiency gains predicted by the analysis, and produces controlled action execution.

1. Introduction

One chilly evening two people, a boatman and a plainsman, came upon two long narrow pieces of wood on the shore. The plainsman, 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 plainsman 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.

As this story illustrates, our perceptions are affected by our knowledge, by what we are thinking about, and by what we are doing. In turn, what we know, what we are thinking about, and what we do is affected by our perceptions. This paper is concerned with this interaction between reason, action and perception. It takes the position that intelligent behavior can be demonstrated and efficiently executed using symbolic Artificial Intelligence techniques by "interweaving" these three activities into a more tightly integrated, fine grained system as illustrated in Figure 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 sections which describe the design and explore the effectiveness of an experimental system which implements this idea in a specific application area: autonomous mobile robot navigation.

[1] This work has been supported in part by DARPA contract DCA 76-89-C-0017, NSF CII grant CDA-8922572, and by grants to Mount Holyoke College from the Bristol Myers Squibb Foundation, the Howard Hughes Medical Institute Undergraduate Biological Sciences Initiative Program, and the W. M. Keck Foundation.

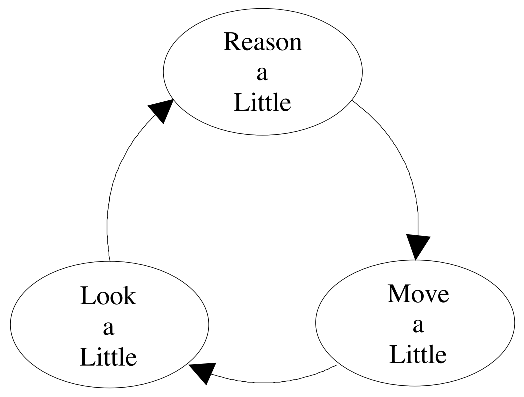


Figure 1 This paper introduces the concept of a fine grained interweaving of reasoning, action and perception. Each process is done on an *incremental* basis, doing only what is necessary to perform the next.

1.1. Statement of the Problem

The problem domain selected for investigating the interaction between planning, action, and perception is that of controlling a mobile robot in performing intelligent tasks. This domain is well suited to the goals of this work 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 might be entered via speech or natural language statements, such as

"There is a red book in Allen Hanson's 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.

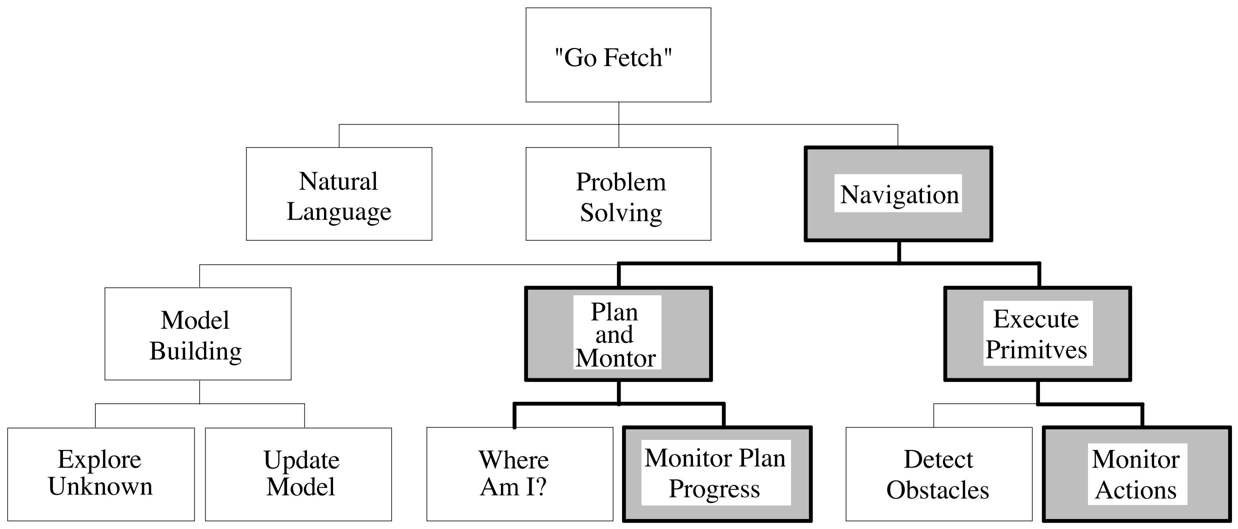
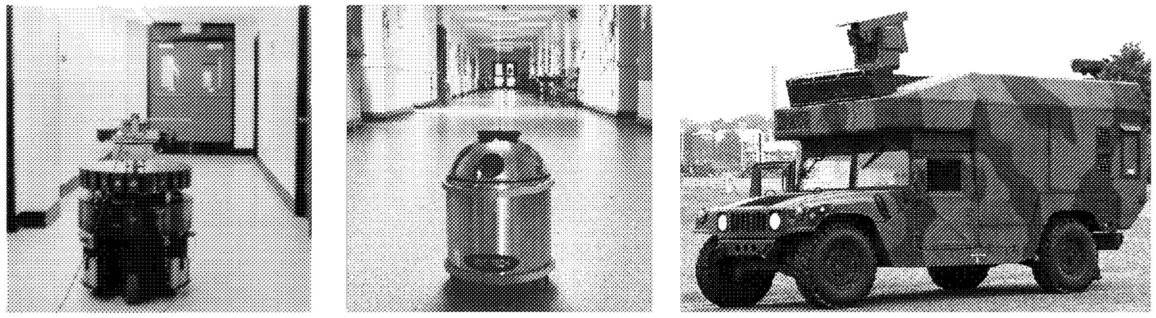


Figure 2 In designing the experimental navigation system it has been assumed that the "Go Fetch" problem can be decomposed according to the above diagram. The paper describes working version of an RML system that implements the shaded modules.



(a) (b) (c)

Figure 3 Experiments have been performed using three different mobile robots: the UMass Denning robot "Harvey" (a), the Mount Holyoke mobile robot "Susan B." (b), and the UMass HUMMWV (c). All are equipped with a black and white TV cameras.

It is assumed in this work that this problem is decomposable; Figure 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 to be 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 [34] on conceptual dependency. The sentence "Go get it!", for example, would result in the goal

```
(and
    (ptrans claude-fennema's-office allen-hanson's-office)
    (atrans book from allen-hanson to agent)
    (ptrans allen-hanson's-office claude-fennema's-office)
    (atrans book from agent to claude-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 claude-fennema's-office allen-hanson's-office)
```

to the navigation module. It is this navigation subproblem that has been chosen as the *initial domain* for investigating the effects of interweaving reason, perception and action.

Navigation uses vision for several *purposes.* The bottom row in Figure 2 identifies the vision capabilities required by a complete navigation system. The system and analysis described in this paper does not yet address all of these capabilities. Although preliminary work has been done in the use of vision for the "update model" [26] and "Where AM I?" [21] problems, the results described here refer to the navigation subsystem highlighted in Figure 2.

Three different robots have been used in this work. Two of these robots, "Harvey" (Figure 3 (a)) and "Susan B." (Figure 3 (b)) have been used for with indoor experiments at the University of Massachusetts and at Mount Holyoke College (MHC), respectively. The third, a HUMMWV (Figure 3 (c)), has been used very recently to perform outdoor experiments with perceptual servoing. These experiments were performed in an open field at the University of Massachusetts. "Harvey", is a Denning mobile robot equipped with a black and white 512 x 512 TV camera, connected via an umbilical cord to a minicomputer. "Susan B." is a smaller, more erratic, but functionally similar robot, equipped with a 640 x 480 black and white TV camera. The HUMMWV was a military ambulance equipped with a variety of black and white and color cameras.

1.2. Related Work

Historically, Artificial Intelligence researchers have divided their investigation into separate studies of reason, perception and action. This phenomenon is very clear in the early robot programs, beginning with the SRI "Shakey" robot project (Nilsson [30]) and the Stanford Cart (Moravec [29]), but it is still evident in more recent systems such as MAUV (Herman and Albus [20]), Carnegie Mellon's NAVLAB (Thorpe, Hervert, Kanade and Shafer [35]), the University of Maryland

System (Waxman, LeMoigne, Davis et al. [37]), AuRA (Arkin [2]), and the Martin Marietta ALV (Turk, Morgenthaler, Gremban, and Marra [36]). A close analysis (Fennema [13]) of these systems shows each of them is an instance of the architecture depicted in Figure 4 (a) and that information flows in these systems as shown in Figure 4 (b). It is characteristic of these systems that each invocation of a module makes a large contribution to the ongoing "intellectual process" of the system. Typically the perceptual (vision) system *completely* analyzes a scene and the reasoning (planning) system develops a *complete* plan. Interaction between modules in these systems is coarse grained. This type of architecture will be referred to as the macromodule architecture .

Rod Brooks led a departure from this pattern with his "layered architecture" (Brooks [6]). His architecture takes the form of a more integrated system of simple processing units that, in a very real sense, looks like a neural net. Unlike the macromodule systems, the interaction between the modules neural nets is fine grained in the sense that each module invocation contributes very little to the overall "intellectual process". Brooks' system and neural net approaches (such as Pomerleau [32]) have been very successful at emulating low level behaviors (e.g. wandering) but, although Minsky [28] has suggested some ideas which might lead to higher level behaviors, these systems have not yet "evolved" to the point where they can exhibit "higher level" behaviors (e.g. planning).

Another departure from the historical, or conventional, approach has considered action and perception together. This group consists of the *active vision* and *purposive vision* communities. The active vision group (e.g. Bajcsy [4]) performs actions as part of their vision processes in order to acquire new information that about the scene. The purposive vision community (e.g. Aloimonos, et al. [1]) uses the knowledge of the actions they will perform to define a set of simple vision modules. In practice both of these systems tend to be coarse grained, macromodule systems.

By the late 1980's a number of researchers began presenting their ideas for new architectures [2], [3], [9], [10], [16], and [31]. These principle problem for these designs to address was that planning, perception and action are computationally expensive operations. To achieve real time performance it was clear that processing had to be distributed or somehow simplified. Some researchers made the observation that more interaction between the modules would be beneficial. The architecture plans for the CMU AMBLER [3] was one step in this direction. Their plans proposed that perception, planning, and motion be separate modules that would communicate by sending messages via a central control module. The role of the control module was to accept messages from the other modules, to decide how to attend to a variety of goals and to decide what to do if things were not going according to plan. It would, for example, inform the planner that its plans are failing or no longer applicable. Payton, Rosenblatt and Keirsey [31] observed that fine grained interaction was key. They designed a system consisted of a collection of fine grained behaviors, designed using connectionist ideas, guided by what they called internalized plans. Internalized plans, as they characterize them, are like gradient fields that, at each point, indicate the best direction toward a goal.

This paper describes another approach, one that interweaves perception, reason and action into a *fine grained, single*

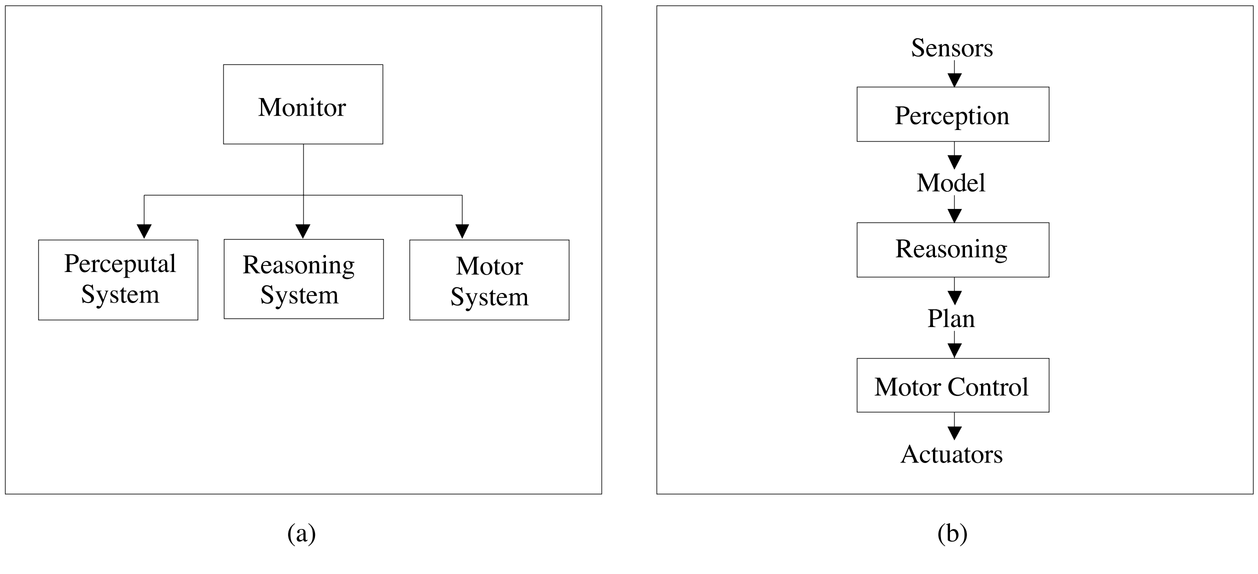


Figure 4 Historically, robot systems separate perception, reason and motor functions into distinct, nearly independent subsystems (a). Typically, the information flow between these modules is as shown in (b).

process architecture that has been given the name "RML" (for "reason a little", "move a little", "look a little"). In the RML architecture reasoning, action and perception are all done *incrementally and depend on each other in a fine grained fashion*. A detailed initial design for the RML system described in this paper was presented in 1988 [10]. At the time of this report, a system has been constructed, analyzed and been used to perform enough experiments to characterize its performance. The next few sections describe the most recent version of this architecture, show that the incremental nature and fine grained interaction reduces the overall computational complexity of the system, and report experiments that indicate how it performs in action.

1.3. Outline of the Paper

The remainder of the paper describes the RML implementation used in this investigation, the complexity analysis, the experimental results, and conclusions which can be made at this time. Section 2 outlines the design of an RML system. It provides an overview of the representations, control structure (plan and monitor), planning process (plan sketching) and "perception" (perceptual servoing). The complexity analysis of the system operations, presented in Section 3, shows that reducing the interaction grain of the system (implementing an RML architecture) reduces system complexity. Section 4 adds empirical evidence to this claim, showing that the complexity gains indicated by the complexity analysis are achieved in typical situations. Section 5 then summarizes the conclusions and discusses work in progress.

2. An RML Implementation

The first detailed design of an RML system appeared during the fall of 1988 in Fennema, Riseman and Hanson [10] and early experiments have been described in Fennema and Hanson [11], [12]. A description of the first complete implementation can be found in Fennema [13]. This section summarizes the key elements of that system.

The implementation is built around a central model that contains information about the *environment*, the robot's *capabilities*, and the *task* to be accomplished. The control structure follows the ideas illustrated in Figure 1. Beginning with a goal the system reasons about a course of action, *doing only what reasoning is necessary* to decide what incremental move to make next. During this *reasoning increment* the system also predicts where it will be and how its perceptions should change as a result of the action. The action increment is then performed and the perceptual expectations are compared with what is seen after the action is complete. 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. Actions are performed in very small increments. Vision in this system does not take on the flavor of an image understanding system or a scene analysis system. The visual sensor is used in a way which is quite analogous to the way a blind man uses his cane. The robot is viewed as choosing its actions based on reasoning done with its model and using its visual sensors to gather evidence that its model and "reality" are in agreement.

2.1. Representations

The system has explicit models of the environment, the capabilities of the agent and the task to be performed. The *environment* is represented as a network of spatial entities, organized in a way which collects relevant geometric, topological and physical properties together (Figure 5 (a)). The specifics of this organization are used to focus reasoning processes. The *capabilities* of the agent are described in terms of the properties of its receptors and actuators. Receptors are modeled in terms of how they transform the environment into sensory patterns and actuators are modeled in terms of what physical changes are expected to result from their invocation (Figure 5 (b)). Finally the *task* at hand is modeled as a plan sketch: a *partially expanded* plan of action composed of subgoals and milestones (Figure 5 (c)).

2.1.1 The environment

The environment is represented as a network, called the locale network. Nodes in this network represent geometrical entities: volumes, surfaces, curves and points. The arcs of the network represent relationships between the spatial entities represented by the nodes: structure, inclusion (contained-by) and connectivity. The structure relations represent the geometry of the environment. The inclusion relation organizes the environment into a system of neighborhoods. This relation is used to specify location and to focus reasoning processes. The connectivity relationship describes how the environment is connected. It is used by the planning operations described in Section 2.2.

Unlike the majority of representational techniques, which categorize the environment in terms of free space, landmarks and objects, this network describes the environment uniformly as a collection of volumes, called locales. A locale is any volume which has semantic significance to the navigation problem. It represent the kind of entity referred to by English phrases such as "South Hadley", “Claude’s Desk”, or “The eraser in the top drawer”. Locales also designate the kind of localization indicated by English phrases such as: "... in Amherst", "... on the third floor", "... in the office", or "... in the desk drawer." These specifications are inexact when viewed from a local perspective; two distinct points can be in the same locale. However, from a global perspective these 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. For this reason each locale is equipped with a *local* coordinate system.

Locales are represented by nodes in the network. Their structure is described in terms of their surfaces which, in this implementation, are modeled as polyhedra. The faces, regions, lines and vertices (points) of these polyhedra are also nodes in the network. Faces, in turn, are divided into regions, defined as areas of the face that have uniform perceptual properties. These regions are also nodes in the network. The locales, their faces, the regions, the lines that bound the faces and the regions, and the vertices where lines meet are all connected by network arcs that identify their structural relationships. Figure 5 (a) illustrates structural relationships between network nodes by the arc connecting the locale called "Allen's Office", the face called "East Wall", and the regions of the East Wall.

The inclusion relation defined between locales is the "contained-by" relation. A locale l_2 is contained-by the locale l_1 if l_2 is a subset of l_1 and l_2 is not a subset of any other locale l_3 which is a subset of l_1. This relations organizes locales into a contained-by hierarchy as illustrated in Figure 5 (a) by the arcs that connect the Research Center, Allen's office and the office furniture. In the implementation described in the following sections, each set of sibling locales l_1, ..., l_m in the contained-by hierarchy form a partition of their common parent locale l_P.

The contained-by hierarchy is used for describing position and to focus reasoning processes. A location is represented as a pair (**l**, **p**) where **l** is a locale containing the position in question and **p** is its pose, expressed in the coordinate system associated with **l**. The locale identified in this representation provides some localization in itself. It identifies a subspace containing the location. It also provides reasoning processes with a pointer into the network for access to information about

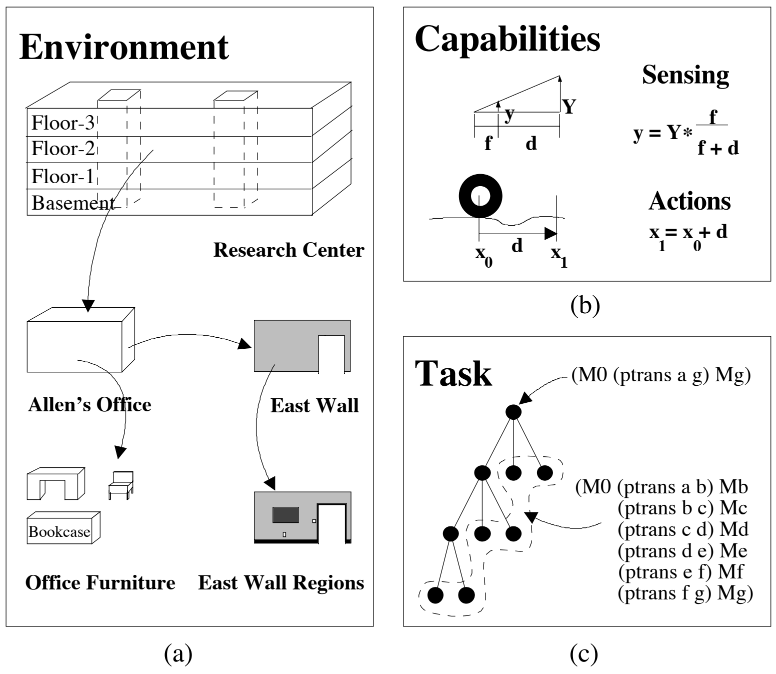


Figure 5 The RML system represents the environment as a network (a). The nodes in this network represent spatial entities: volumes, surfaces, lines and points. The arcs represent relations: inclusion (contained-by), structure, and connectivity; the capabilities of the agent’s sensors and effectors are modeled in terms of how they transform or affect the environment , respectively (b); and the task is represented in terms of goals and milestones (c).

that location. The use of local coordinate systems addresses the problems that limited accuracy presents when working with a global coordinate system.

The focusing mechanisms use the contained-by hierarchy to limit the volume of space and the level of detail considered. The scope mechanism, limits reasoning to facts about a particular subspace of the environment by selecting a locale, called a scope-locale, and focusing only on the information in the subtree it defines. Another mechanism, the perspective mechanism, limits the amount of detail considered by limiting how deeply reasoning probes into that subtree. When reasoning about a trip from Mount Holyoke to the University of Massachusetts, these mechanisms provide means to ignore facts about Saturn (scope) and the contents of the box in a drawer of the spare desk in the hallway (perspective).

The connectedness relation is used by the planning processes while generating paths. This relation has been implemented as a pointer associated with each face that is not a barrier to travel (e.g., a doorway). It identifies the locale that shares that face.

Finally, reasoning activities require knowledge about the structure and appearance of locales. Associated with each face is a set of properties that identify it as a "barrier" or as a "door-way" for navigation purposes. Furthermore, as mentioned above, each face may be partitioned into areas of uniform perceptual properties called regions[2]. Surface reflectance $\mathbf{r}$ ($0 \leq \mathbf{r} \leq 1$) was the property used to define regions in the implementation. Figure 5 (a) shows one of the faces of Allen's office, the "East Wall", and the regions used to describe its appearance. Accessing these planning and perceptual properties can be done very quickly by traversing only a few arcs.

2.1.2 Capabilities

The processes that reason about routes and perception use a model of the interface between the agent and its environment. This model is used to predict the consequences of taking an action and to interpret the input from its sensors. In its current form it is fairly straightforward.

The only sensors used in work with Harvey, Susan B., and the HUMMWV were TV cameras. These cameras were modeled as devices that project the environment onto an image plane according to the laws of simple geometric optics. This model is implemented as a set of procedures that perform perspective projection, clipping and other transformations based on a set of parameters (lens focal length, inter-pixel distance, "retina" dimensions, camera pose, etc.)

The agents ability to perform actions is modeled in terms of primitive actions and their predicted effect on the environment. 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. For a complex agent, such as a human, the number of primitive actions can be very large. The agents used in this work, Harvey and Susan B., are relatively simple. Their motions can be expressed in terms of two primitive actions:

Primitive Action	Predicted Effect
(move distance)	The agent the specified distance forward (or backward) along a straight line in the direction of the current heading. The agent heading remains unchanged. There is no change of locale.
(turn angle)	The agent's heading is changed by the specified angle. The agent's location is unchanged.

Any action Susan B. and Harvey are capable of performing can be decomposed into a sequence of these two primitive actions. As will be discussed in Section 2.4, the model for the effects of these actions would be very much in error if they were executed without perceptual feedback. For this reason, these primitive actions are carefully controlled in an RML loop called action-level perceptual servoing. Action-level perceptual servoing accomplishes these actions accurately by interweaving incremental actions with vision. These incremental actions, (agent-move distance) and (agent-turn angle) have the same models

Incremental Action	Predicted Effect
(agent-move distance)	This action is assumes the model used by (move distance) but only for distances less than 1 foot. The effect for larger distances is considered unpredictable.
(agent-turn angle)	This action uses the (turn angle) model but only for angles less than 10 degrees. It is assumed that larger angles will result in unpredictable results.

The details of action-level servoing will be discussed in Section 2.4.

[2] Each face may have two sets of regions associated with it: one associated with the inside surface of the face, the other with the outside.

2.1.3 The task

The RML architecture forces some changes in how a task is represented. Since reason, action and perception are carried out incrementally it is necessary to represent partial solutions in a meaningful way. In addition, since these activities depend on each other, we need a way to represent the information needed by each process and how it relates to the task at hand. The solution to this problem turns out to be very simple. Tasks are in terms of goals and perceptual milestones.

Goals express desired accomplishments in terms of the known spatial entities described in the locale network. Two types of goals used: ptrans goals and mtrans goals.[3] A "ptrans" goal, written (ptrans L_1 L_2), refers to a physical change in the robot's location from L_1 to L_2. An "mtrans" goal, written (mtrans L_1 L_2), specifies a "mental" change. In this case L1 and L2 refer to the same "place", but in terms of different locales. An "mtrans" goal invokes a change of coordinate system.

In the RML architecture plans are developed incrementally. The plan develops as action takes place. It is not complete until the goal is achieved. To represent developing plans, the notion of a plan sketch is introduced. A plan sketch is a sequence of subgoals, some of which may not be primitive, that together solve the original goal. This concept will be discussed in more detail in Section 2.3.

The term milestone, as used in this paper, is borrowed from project planning literature (for example, Brooks [5]). It is a measurable event which can be used to determine that execution of a plan has reached a certain stage. In this work, the term milestone refers to a measurable relationship that should exist between the agent and one or more landmarks after the accomplishment of a goal. Milestones that can be verified by perceptual means and are called perceptual milestones. In the following sections, milestones identify three dimensional entities selected from the model and indicates how they should relate to the agent when a subgoal has been accomplished. This selection process will be described in more detail in Section 2.4.

Tasks in the RML system are represented as a sequence (M_0 S_1 M_1 S_2 M_2 ... S_F M_F) of subgoals S_k and milestones M_k that satisfy the conditions:

1. The sequence (S_1 ... S_F) is a plan sketch,
2. Each M_k is a milestone that should be recognized after the goal S_k is achieved, and
3. M_0 is a milestone that must recognized for the plan sketch (S_1 ... S_F) to be valid.

It is the explicit addition of milestones that makes this structure novel.

In the remaining sections of this paper the phrases "task representations" and "plan sketches" will be used interchangeably. The close relationship between task representations and their embedded plan sketches should keep this from being confusing.

RML reasoning begins with an initial task description (M_0 (ptrans a g) Mg), representing the original goal. Reasoning increments change the task description by refining it and by removing recognized milestones and accomplished subgoals. The example in Figure 5 (c) shows a depth first refinement of the initial plan sketch (M0 (ptrans a g) Mg). The resulting plan sketch, represented by the leaf nodes of the graph in the figure is

(M_0 (ptrans a b) Mb (ptrans b c) Mc (ptrans c d) Md (ptrans d e) Me (ptrans e f) Mf (ptrans f g) Mg).

Sections 2.2 and 2.3 describe plan sketch refinement in detail.

2.2. Plan and Monitor

The control structure of the RML system is implemented by an algorithm called plan-and-monitor. As will be discussed in Section 2.4, this algorithm implements the RML loop at three nested levels. The "plan-and-monitor" algorithm is illustrated in Figure 6. This process begins with a plan sketch

($M_{\text{robot-lab}}$ (ptrans robot-lab ted's-office) $M_{\text{ted's-office}}$)

representing the initial goal. If the first milestone is not recognized, the goal is invalid and an error is returned. Otherwise it is removed and plan-and-monitor is invoked. It begins by invoking the sketch-a-plan process (described in Section 2.3) which refines the first subgoal of the plan sketch. The goal, for example, might be decomposed into the plan sketch

```
( (ptrans robot-lab research-door-1)                 M_research-door-1
  (ptrans research-door-1 engineering -door-3)       M_engineering-door-3
  (ptrans engineering -door-3 ted's-office)          M_ted's-office )
```

[3] This interpretation of mtrans and ptrans is consistent with the work done by Schank and Ableson [34]

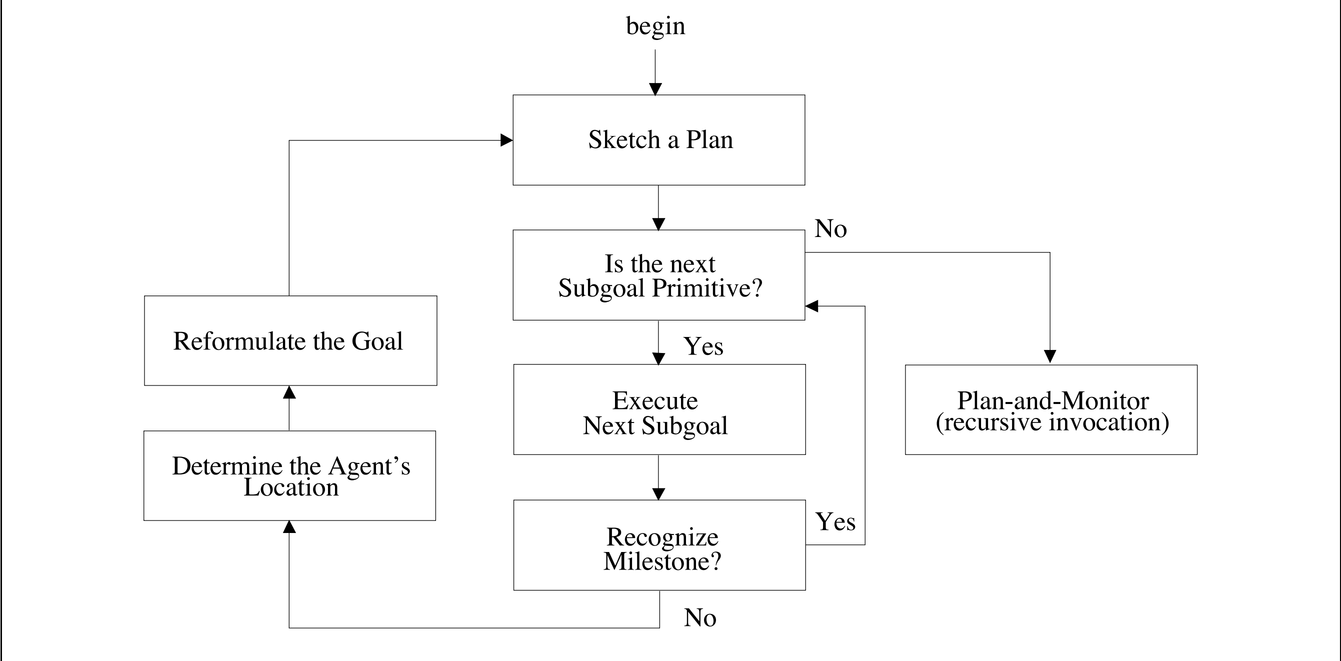

begin
Sketch a Plan
Is the next Subgoal Primitive?
No
Yes
Reformulate the Goal
Execute Next Subgoal
Plan-and-Monitor (recursive invocation)
Determine the Agent's Location
Recognize Milestone?
Yes
No

Figure 6 The process of interweaving reason, perception, and action is orchestrated by an algorithm called "plan-and-monitor", summarized here as a flowchart.

as illustrated in Figure 7 (a).

Plan-and-monitor then considers the first subgoal of this sketch. If this subgoal 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 7 (a) the first subgoal

(ptrans robot-lab research-door-1)

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

((ptrans robot-lab research-center-elevator-second-floor)	$M_{research-center-elevator-second-floor}$
(ptrans research-center-elevator-second-floor research-center-elevator-first-floor)	$M_{research-center-elevator-first-floor}$
(ptrans research-center-elevator-first-floor research-door-1)),	

illustrated in Figure 7 (b). This process of plan refinement continues, recursively refining the first step of each plan sketch until the first subgoal is a primitive one. This is the situation shown in Figure 7 (d). The result of this process is a depth first refinement of the goal into a decomposition tree like the one shown in Figure 5(c). The leaves of this tree form the current working plan sketch :

((ptrans laboratory bench-corner)	$M_{bench-corner}$
(ptrans bench-corner desk-corner)	$M_{desk-corner}$
(ptrans desk-corner laboratory-door)	$M_{laboratory-door}$
(ptrans laboratory-door elevator-Floor-2)	$M_{elevator-Floor-2}$
(ptrans elevator-Floor2 elevator-Floor-1)	$M_{elevator-Floor-1}$
(ptrans elevator-Floor-1 research-door)	$M_{research-door}$
(ptrans research-door engineering-door)	$M_{engineering-door}$
(ptrans engineering-door ted's-office)	$M_{ted's-office}$)

The first three subgoals in this sketch are primitive. The others must be decomposed before they can be acted upon. Plan sketch detail is developed only as required. This keeps plan reformulation costs low, should any replanning be required.

As soon as the reasoning process has generated a plan sketch whose *first subgoal* is primitive, action begins. An attempt will be made to accomplish it. Primitive subgoals are accomplished using a simple version of the RML loop called action-level perceptual servoing. This process improves the accuracy of primitive action execution and increases the likelihood that they will be properly carried out. Action-level perceptual servoing is described in detail in Section 2.4.

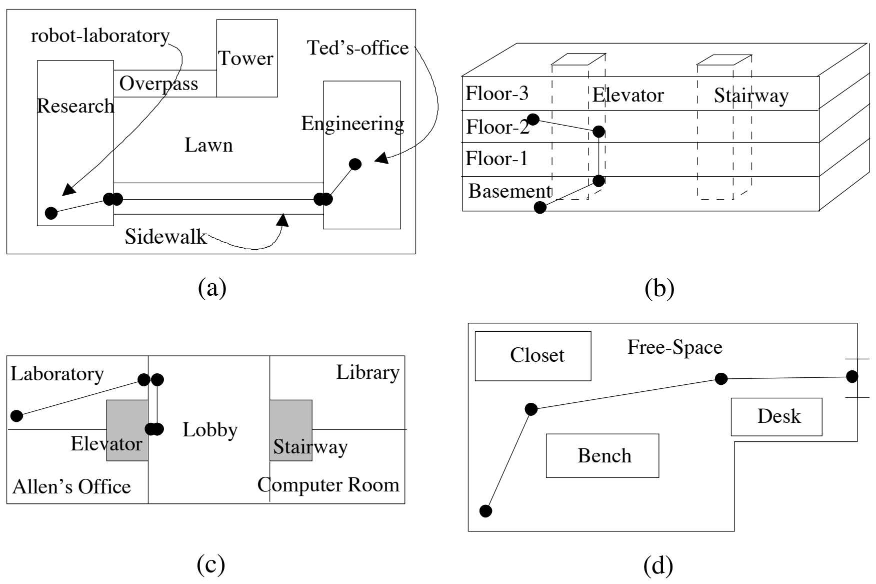


Figure 7 Plan and monitor develops a plan sketches by refining goals in a depth first fashion. In (a) the goal of moving from the robot laboratory to Ted Djaferis' office has been decomposed into three subgoals. Illustration (b) shows the first subgoal of (a) further refined. This process continues in (c) and (d). In (d) the first subgoal is a primitive one so motion can begin.

When the subgoal execution is complete (or "thought" to be complete), an attempt is made to recognize the milestone associated with it. The landmarks of the milestone are identified and their relationship to the agent is determined. If the relationship is within tolerances the milestone is said to be "recognized". This is an indication that reason (as reflected in the plan sketch) and reality are consistent so the next subgoal is considered. Milestone recognition is described in greater detail in Section 2.4,

If the milestone is *not* recognized then the agent is lost. In this case the agent must determine its location, formulate a new goal, and construct a new plan sketch. There is work in progress at Mount Holyoke and at the University of Massachusetts which addresses the process of determining the agent's location when it is lost (Where Am I?). As mentioned in Section 1 (Figure 2), however, this is beyond the scope of this paper. Once the location of the robot has been determined, however, plan-and-monitor would be reinvoked with the goal (ptrans newly-found-location old-finish-location).

2.3. Plan Sketching

The plan-and-monitor algorithm develops plans incrementally using a process referred to as "plan sketching". This section describes an algorithm, sketch-a-plan, which uses the scope and perspective mechanisms mentioned in Section 2.1 to implement a version of that process.

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[4]:

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

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

[4] For simplicity, milestones will usually be omitted. This should cause no confusion.

((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 uses a modified version the A* algorithm (Hart, Nilsson and Raphael [17], [18]). The modification takes advantage of the representation of space described in Section 2.1 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 Section 2.3.3. First, however, it will be helpful to describe how A* has been used in its implementation.

The A* algorithm, which originally appeared in Hart, Nilsson and Raphael [17], is a graph search algorithm that has often been used for path planning (for example by Raphael [33], Lozano-Perez and Wesley [27], and Arkin, [2]). 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 [17], [18]).

A* is efficient by these standards, but its growth characteristics tend to be exponential in the number η of nodes. The reason is that, 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 to determine whether or not it is reachable and whether or not is in the set of open nodes. For a path of length ρ this means that A* has a worse case complexity of $O(\eta^{\rho})$. This exponential growth is characteristic of state space search based planning algorithms (Hendler, Tate and Drummond [19]).

There are two factors that strongly influence how efficiently A* can perform on a given problem:

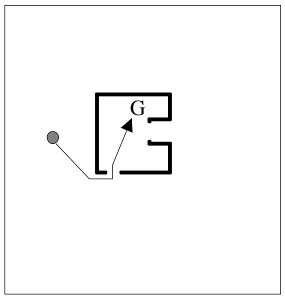
1. The way the problem is interpreted as a graph. This has a strong influence on plan generation efficiency because it determines η. η should be kept as small as possible, without sacrificing plan quality.
2. The design of the successor function. The successor function should choose nodes in a focused manner. It should return as few nodes as possible without sacrificing plan quality.

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 be used to control the complexity of A* by providing several focusing mechanisms. The resulting complexity will be discussed in Section 3.

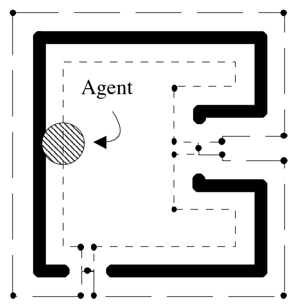
2.3.1 The graph and the heuristic function

The state space graph used by sketch-a-plan is motivated by the often used space described by Lozano-Perez and Wesley [27]. In this scheme, obstacles, which are modeled as polygons, are "grown" into the surrounding free space by one half the width of the agent. The vertices of these new polygons become planning nodes and arcs connect vertices if there is a straight-line 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 as locales makes some modification necessary. Since it is possible in this system of representation for the agent to sometimes be inside a locale and at other times be outside of it (Figure 8 (a)), each locale is assigned two planning boundaries: one inside the locale and the other outside (Figure 8 (b)). These planning boundaries are formed by "growing" the boundary of the floor of the locale outward, to form the outside planning boundary, and inward, to form the inside planning boundary. It is from these planning boundaries that planning points (planning nodes) are selected. 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 8 (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 associated with the locale. 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.



(a)



(b)

Figure 8 The representation of space as locales incorporates the idea that a locale can at times be an obstacle and at other times be the environment (a). For this reason each locale has associated with it three sets of planning locations. The first set consists of the points marking the middle of each doorway of the locale. The other two sets are determined by growing the walls of the locale inwards and outwards by one half of the robot's width. The convex vertices (viewed from outside the locale) of the outer curve, shown as a dashed line in (b) form the second set, which is used to plan when the locale is viewed as an obstacle. The third set is formed from the convex vertices (viewed from the inside) of the inner curve, shown as the dotted line in (b). These are used when planning inside the locale.

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 [2]). Incorporating such ideas into the system described in this paper would be a straightforward undertaking.

2.3.2 The Successor Function

The choice of a successor function is of particular importance because it has the potential of keeping planning computation costs low. The choice of successors to a node in sketch-a-plan is focused by three mechanisms: the scoping mechanism, the perspective mechanism and a semantic filter. The scoping and perspective mechanisms, described in Section 2.1, 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 is implemented as a part of the sketch-a-plan algorithm. It determines the smallest locale, called the scope-locale, in which a plan sketch for the specified goal can be constructed. The successor function limits its search for successors to a subtree of the contained-by hierarchy with this locale as its root. This limits the *volume* of the space considered while reasoning about paths. The perspective mechanism, which is part of the successor function, limits how deeply reasoning probes into the subtree defined by the scope-locale. As *currently implemented*, the successor function limits its choice of successors to those associated with the scope-locale and its immediate offspring .

Semantic filtering is also performed by the successor function. As described in section 2.3.1, the planning points associated with each locale 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 navigation problem under consideration offer 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 9 (a)). 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 9 (b)). 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 9 (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 9 (d)). In this case successors will consist only of the doorway-points of the sublocale L.

5. S is a door-way-exit (Figure 9 (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 obtained using the connectivity relation described in Section 2.1. This point is, by definition, reachable whenever the door is open.

This successor function accepts two arguments, a location and the scope-locale. From the set consisting of G and the planning points associated with the scope local and its immediate offspring it returns those selected by the semantic filter. This algorithm focuses A* reasoning to the local environment and makes RML planning very efficient.

2.3.3 The Sketch-a-Plan Algorithm

The sketch-a-plan algorithm is relatively straightforward (see Figure 10). Given a goal:

(ptrans start-location finish-location)

sketch-a-plan first chooses a scope-locale. The first approximation to this locale is the smallest locale that contains both start-location and finish-location. A* is then applied using the successor function described above. If A* is successful in finding a path, it is returned as the plan sketch. It is possible, however, that A* will be unsuccessful if this scope-locale is too small to contain a free path. In this case A* must be reinvoked using a larger scope-locale. If the current scope-locale is the universe, this is not possible and sketch-a-plan fails; otherwise the parent becomes the new scope-locale.

When A* fails in its first attempt to find a path and a path does exist, it means the locale network is not "properly" described. A locale network properly describes an environment if the free space in each of its locales is connected. When the free space in a locale is not connected, the network can be repaired by splitting the locale into sublocales and performing some minor network reorganization. This is the process referred to as restructuring in Figure 10. Details about restructuring can be found in [15].

Once restructuring is complete, planning with A* invoked using the new scope locale. Each time A* fails to find a solution sketch-a-plan continues to work 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).

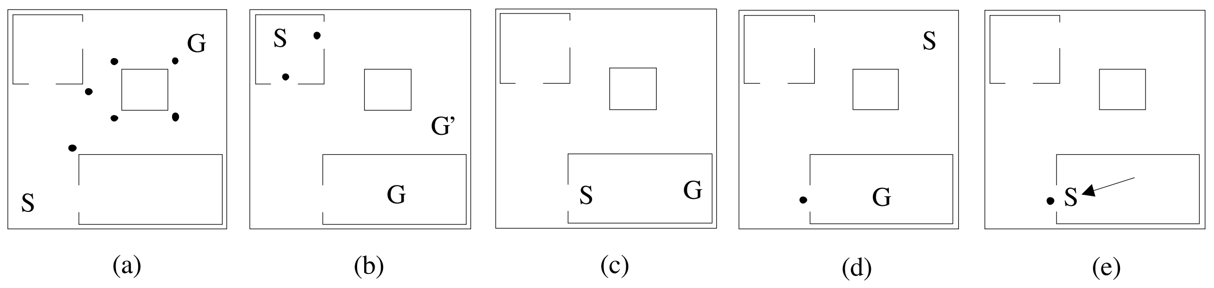


Figure 9 Successors to a start location, S, are determined in part by the situation. These situations are illustrated here, using the example of a locale L with four sublocales: free-space, two buildings and a fountain. In (a) both S and the goal location, G, are in the *free space* of L; in (b) S is in one of the buildings (not in the free space of L) 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 sublocale; in (d) S is in the free space of L while G is not; and (e) represents the situation where S is the result of a move with intent to pass through a door.

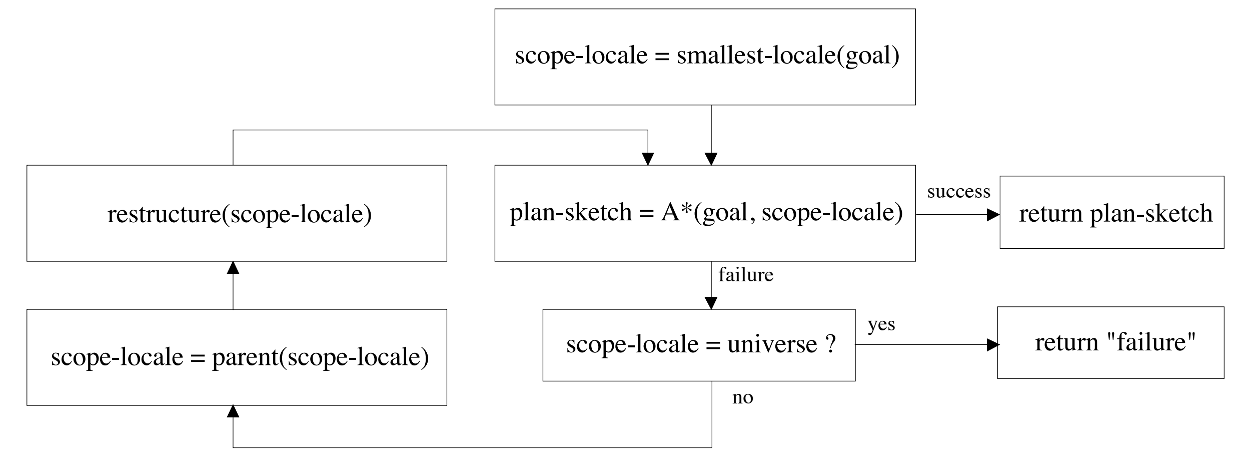

scope-locale = smallest-locale(goal)
restructure(scope-locale)
plan-sketch = A*(goal, scope-locale)
success
return plan-sketch
failure
scope-locale = parent(scope-locale)
scope-locale = universe ?
yes
return "failure"
no

Figure 10 The sketch-a-plan algorithm.

The way that sketch-a-plan selects its scope-locale, together with the way that the successor function is implemented does not guarantee that the plans generated will be optimal. It is easy to construct cases where the plan generated by the RML system will not be optimal. We need to keep in mind, however, that the path *executed* by an agent moving in a real environment will perhaps never be optimal and, due to surface unevenness, wheel slippage, and errors in the agents sensors, will usually be different from what is planned. It is sufficient that plan quality be high. Experiments with "typical situations" show RML plan quality to be high (see Section 4).

2.4. Perceptual Servoing

As mentioned in Section 2.2, plan-and-monitor has the effect of controlling the agent's behavior with a three level servo loop (Figure 11). These three levels are called action-level, plan-level and goal-level perceptual servoing. Each of these levels is an example of the RML paradigm.

The reasoning portion of these perceptual servoing loops does just enough reasoning (reason a little) to decide what *incremental* action to take next and to construct appearance predictions for landmarks that should be perceived after that action is complete. This reasoning is based entirely on what is in the representations. The incremental action is then performed (move a little) and a new image is acquired. The appearance predictions for each landmark are then projected onto the image plane and are 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.

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. Implementation of goal-level perceptual servoing is work in progress. It is discussed briefly in Section 2.4.6.

2.4.1. Selecting Landmarks

In principle, a landmark can be any 3D entity: an object, a group of objects, a group of lines, 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 distinguished from other 3D entities in an image using correlation. Landmark selection is a reasoning step, based solely on what is in the representations. The process analyzes the vertices in the locale occupied by the robot and selects those which are surrounded by regions which differ in visual properties.

Distinctive reflectance patterns occur where regions of differing reflectances meet. Particularly useful patterns occur where the boundary common to those regions curves sharply. In the current representations this happens at vertices (Figure 12 (a)). Selecting landmarks is done by searching the locale network for vertices that are common to two (or more) regions that differ in reflectance. The structure of the network makes this computation relatively straightforward and efficient. The

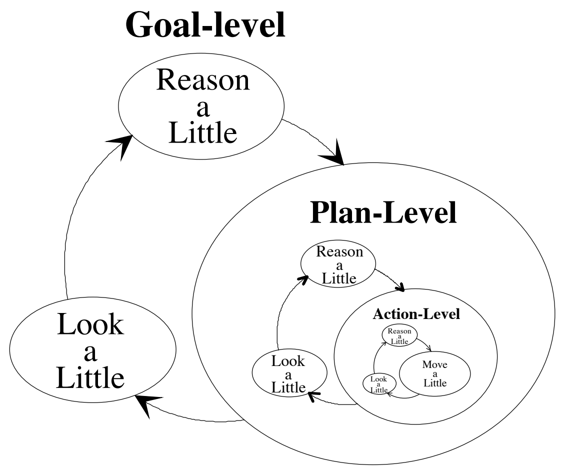


Figure 11 Achieving navigational goals is the result of three levels of perceptual servoing: action-level, plan-level and goal-level.

search is focused using the contained-by hierarchy. The scope locale in this case is the locale containing the agent. This is readily available from the representation of the agent's location.

agent-location = (locale, pose)

As with plan sketching, the current implementation limits this search to the vertices associated with the scope locale and its offspring (freespace and the objects contained in the locale). These locales 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 scope 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. Let L = the set of all vertices associated with the regions on the inside surface of the locale and the outside surface of each offspring locale.
2. Delete from L all vertices which are not expected to be visible to the agent from the location it will be in after the motion increment. This is accomplished by first clipping the projection to the image plane (ignoring occlusion) and then deleting occluded vertices from what remains.
3. Delete from L all vertices which are not part of the common boundary of at least two regions that differ in reflectivity by some threshold.
4. Return L.

Each vertex returned by this procedure identifies a surface patch defined by the visible 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.

2.4.2. Constructing Appearance Templates from the Model

An appearance template is an image array which specifies how a landmark should appear in the image. For the vertex landmarks selected by the procedure described in the previous section, these are n x n image arrays[5]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 (see Figure 12 (b)). 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. Let **R** = the set of all regions which have the vertex **v** on their boundary.

[5] The arrays that have been used are very small (6 x 6).

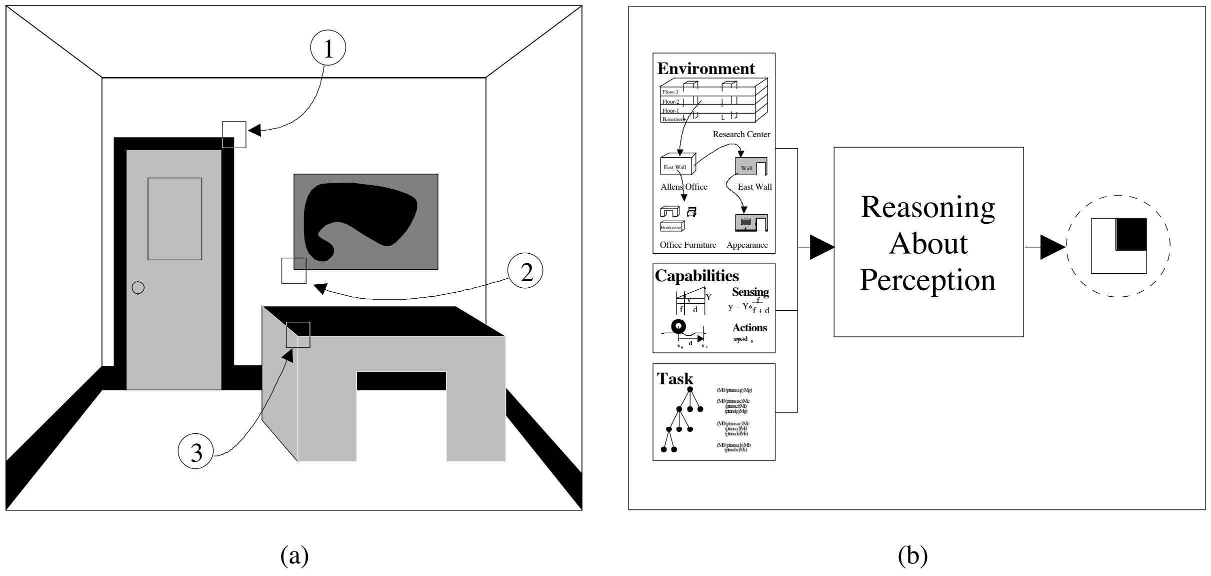


Figure 12 Landmarks used in the experiments described in this paper surface are patches associated with vertices in the locale. The areas surrounding vertices (1), (2) and (3) in (a) are typical examples. The appearance templates used in action level and plan level servoing are constructed by ray tracing, using information from the locale network and the task model (b).

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 in **R**.

   If the ray does not intersect any of the regions in **R** one of two possible situations is likely. Either the pattern of regions associated with the vertex is 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. In either case, use of the landmark with the matching scheme described in Section 2.4.3 will produce unreliable results. A null template is returned and the landmark will not be used.

   Otherwise assign to the pixel the value 255***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 templates constructed by this process are used to match the landmarks with their projections in an image. This matching is done using correlation in a way that is based on reflectance values, rather than intensities, to reduce the ever present effects of uneven illumination.

2.4.3. Matching Appearance Templates to the Data Using Correlation

Finding the location of the image of a landmark in an image is done by finding the best match for the appearance template in a p x q window centered about its expected location (Figure 13). 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, since imaging is a projective transformation.
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

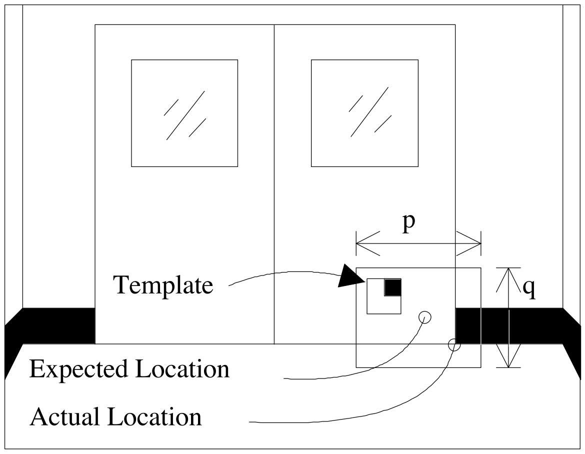


Figure 13 Once the template is constructed from information in the model, both the template and image are transformed as described in the text. The match for the template is then found using normalized correlation on the transformed images over a p x q window.

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 2.4.2.

The third problem is managed by transforming each acquired image P(i,j) to a new image P'(i,j) which is independent of the scene illumination. Constructing a general transform for this purpose would be very difficult. Assuming, however, that the surfaces in the environment are Lambertian, that they can be locally approximated by a plane, and that the illumination incident on the surfaces can be approximated locally by a constant, the image can be approximated by the expression:

$$P(i,j) = c*I*\mathbf{r}(i,j) \quad 0 \le i,j \le n$$

where I is the (constant) light intensity over the surface patch, the constant c represents the fact that the other factors (angle of incidence, angle of reflection, atmospheric absorption) remain relatively constant over the small patch, and **r**(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 P'(i,j) has values given by the formula:

$$P'(i,j) = \frac{P(i,j)}{\sum_{(k,l)\in W} P(i+k,j+l)} = \frac{c*I*r(i,j)}{\sum_{(k,l)\in W} c*I*r(i+k,j+l)} = \frac{r(i,j)}{\sum_{(k,l)\in W} r(i+k,j+l)}$$

where (k,l) ranges over the set $W = \{(k,l) \mid -(w_1/2) \le k \le (w_1/2)) \text{ and } -(w_2/2) \le l \le (w_2/2)\}$ 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.

Matching the appearance templates to image data is done using normalized correlation. First the appearance template T(i,j) and the acquired image P(i,j) are transformed to the intensity independent images T'(i,j) and P'(i,j). Then the transformed template T'(i,j) is matched against the transformed image P'(i,j) using normalized correlation, C(i,j). The value of this function is defined by the equation.

$$C(i,j) = \frac{2*\sum_{(k,l)\in S} P'(k+i,l+j)*T'\left(k+\left(v_1/2\right),j+\left(v_2/2\right)\right)}{\sum_{(k,l)\in S} P'(k+i,l+j)^2 + \sum_{(k,l)\in S} T'\left(k+\left(v_1/2\right),j+\left(v_2/2\right)\right)^2}$$

where $S = \{(k,l) \mid -(v_1/2) \le k \le (v_1/2)) \text{and } -(v_2/2) \le l \le (v_2/2)\}$ is determined by the size (v_1 x v_2) of the template T. The computation of this value is, of course, made more efficient by rearranging the terms in the expression. This correlation method has proven to be very reliable for locating landmark images in the indoor and outdoor environments used for the experiments described in Section 4. It is very tolerant of light level and has been used reliably in a very broad range ambient lighting conditions. Some of these experiments have been described by Fennema [14].

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.

2.4.4. Action-Level Perceptual Servoing

Action-level servoing is used to improve the accuracy of primitive action execution. Primitive actions are carried out incrementally, using the location of landmark images to compute necessary corrections. This RML loop begins by predicting the location the robot will be in after the motion increment is complete. This location is used to select landmarks and to construct appearance templates. The location of the landmarks in the image is then found by matching the templates with data in the image. After the landmarks are matched to the image data their image location is used to compute the error in the heading of the robot. To see how this is done, 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, as can be seen from Figure 14,

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

Now, since tan(b) = (-x/f),

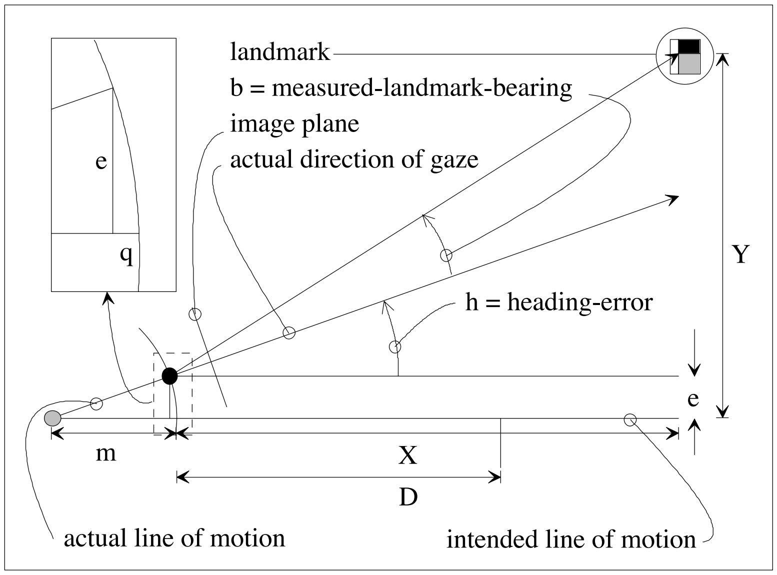


Figure 14 Starting at the location marked by the symbol ● an agent will deviate from its intended line of motion and finish an incremental motion at the location marked by the symbol ●. Experiments have verified that, for very small distances, the path between these two points can be approximated by a straight line. This approximation makes it possible to compute the heading error from the position of a landmark in the image.

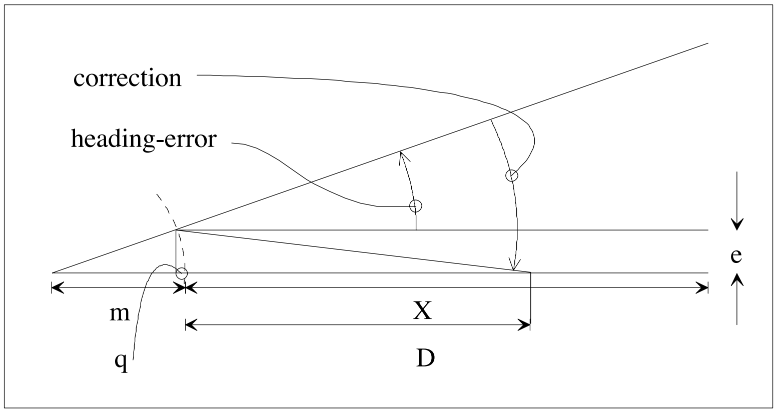


Figure 15 Once the heading error is determined, the robot is turned toward a distant point on its original intended path.

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

Consequently, 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 the experiments performed with Harvey the quantity e was on the order of .02 ft and q was on the order of .05 ft. Hence this approximation is more than reasonable for landmarks more distant than 3 feet.

Given this approximation to the heading error, the corrective action is determined according to the geometry indicated in Figure 15. 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 is determined, using e = m*sin(h) and q = e*tan(h) as follows:

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

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.

2.4.5. Plan Level Perceptual Servoing

The current plan-level perceptual servoing algorithm uses the landmark selection, template construction and matching procedures described in Sections 2.4.1, 2.4.2 and 2.4.3. The resulting matches and the 3D model information are then used to determine the robot's pose, utilizing a 3D pose determination algorithm developed by Kumar and Hanson [24], [25]. If the robot location, as determined by the pose computation, agrees with or is sufficiently close to what is expected then the milestone has been "recognized" and there is no need to replan. Under these circumstances, the first location in the next subgoal in the plan sketch is adjusted to reflect any 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.

If 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.

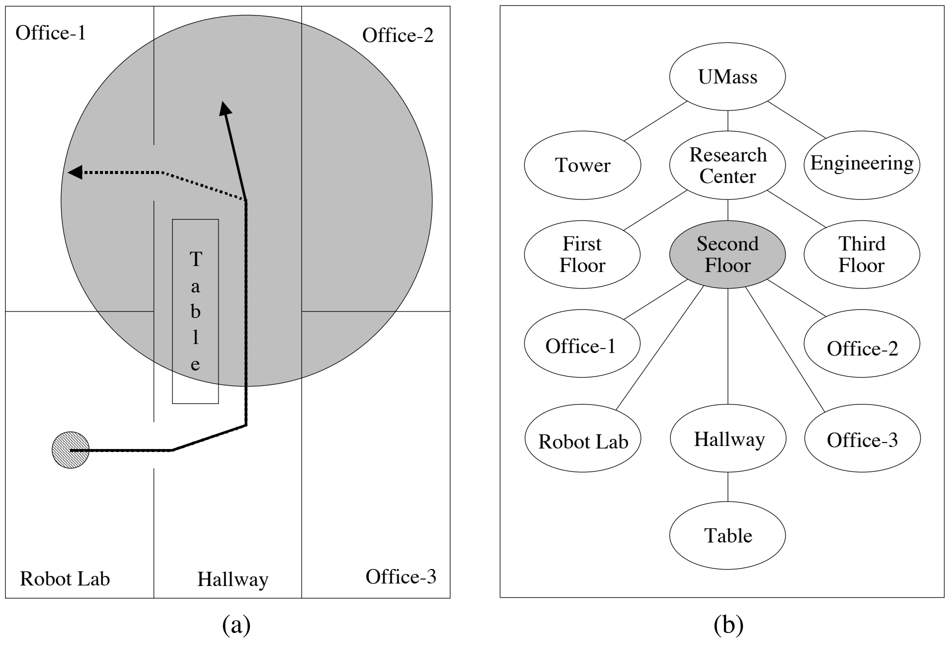


Figure 16 Goal level servoing would be invoked whenever the agent gets lost. In.(a) the agent's goal was to go into Office-1. It followed the plan but at the Office-1 end of the table it made an incorrect turn. This error was not discovered until it attempted a milestone recognition. A strategy for determining the agent's location would be to use the last successful milestone recognition and the distance it has traveled since that time to construct a circle of uncertainty (the large shaded circle in (a)). The smallest locale containing all the locales that intersect this circle (the shaded locale in (b)) defines a subtree in the contained by hierarchy that can be searched to determine the agent's location (see text).

2.4.6 Goal Level Perceptual Servoing

Goal level perceptual servoing, the outermost loop illustrated in Figure 11, is invoked when the agent gets "lost". This is the situation when the agent fails to recognize the milestone associated with a subgoal it "believes" it has just completed. In the plan-and-monitor algorithm (see Figure 6) this is the point where the agent must determine its location, reformulate its goal, and restart the planning process. This corresponds to an experience most of us have had. While following instructions to someone's house (a plan sketch) we accidentally make a wrong turn (see Figure 16 (a)) but continue driving along the new road (action level servoing) until we have gone the distance our friend specified. Now, to our chagrin, we cannot find the big white church (a landmark used as a milestone) mentioned in the instructions. We are lost. We must find out where we are and replan.

Implementation of this portion of the system is in its early stages. It depends on a solution to the "Where Am I?" problem (see Figure 2), which is far from solved. We do envision the locale network, however, as very useful in this regard and have used it in some early investigations. The contained-by hierarchy (Figure 16 (b)) can be viewed as a tree of locales to be searched. It is possible, for example, to use information about the location of the last recognized milestone and the distance traveled to estimate a region that contains the agents current location (the shaded circle in Figure 16 (a)). The least common ancestor of the locales that intersect this region would be the place in the hierarchy to begin the search (the shaded locale in (Figure 16 (b)). The search would progress down the tree to progressively smaller locales, thus improving knowledge of the location of the agent. The search would be implemented in an RML loop, using the knowledge about what can be seen in each locale to reason about what to look for.

One of the first attempts to implement this strategy is reported by Kapur [21]. Kapur generated candidate locales using the kind of reasoning just mentioned. She then determined the locale containing the agent by comparing baseboard points identified in images with those represented locale network. Candidate locales were eliminated when the two sources of information were inconsistent.

Recently, a group of Mount Holyoke students have begun looking at another approach. They treat the locale network as a neural net by associating an activation state with each locale and using the network relations to implement connections.

They have been investigating the use of simple image measurements as inputs to that net. Their hope is for the locale containing the agent to become the most active one.

3. Analytical Results

The claim was made in Section 1 that use of the RML architecture results in substantial savings in system computations. This section supports that claim for the current implementation in the *worst case* situation by presenting analytical arguments[6]. Section 3.1 presents the complexity analysis for RML perception and shows that the cost of perception decreases as the grain size of the RML system decreases. It also develops an expression for the complexity of the total effort necessary to achieving a goal in terms of the total planning effort. Section 3.2 addresses planning complexity and argues that RML style interweaving reduces the complexity of planning.

3.1 The complexity of RML perception.

"Perception" in the RML architecture is very simple. This simplicity has been made possible by the RML architecture and, to some extent, by the nature of the navigation problem. "Perception", as described in Section 2, is used for two purposes: for determining heading errors in action-level servoing and for "recognizing milestones" in plan-level servoing. As mentioned in section 2.4, plan-level and action-level servoing are very similar; both have the following structure:

1. Select Landmarks (Section 2.4.1)
2. Construct Templates (Section 2.4.2)
3. Match Templates to Data (Section 2.4.3)
4. Use this information:
    a) to correct heading while performing action-level servoing (Section 2.4.4)
        or
    b) to adjust the plan sketch plan-level servoing (Section 2.4.5)

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.

A detailed discussion of the complexity for steps 1, 2, and 3 can be found in [13] and [15] but a summary of the results is as follows:

1. The complexity of the *landmark selection* process is a function of the *local* complexity of the environment, as measured by the number of faces m and vertices n associated with the locale containing the agent and that locale's offspring. The total complexity for landmark selection is O(n*m). Both action-level and plan-level servoing procedures use a fixed number of landmarks, regardless of the complexity of the environment. In the following, let n' be this number.

2. The *template construction* process is performed for each of the n' vertices selected as landmarks. Each template is a k x l image array that predicts how that landmark should appear in the image. If we let t be the maximum number of faces that meet at a vertex in the model, the total complexity is O(n'∗k∗l∗t). The number n' of selected landmarks is fixed and the template dimensions k x l must be small for the system to behave properly. It is also safe to assume that t is small, for example t < 5. Thus the complexity is constant O(n'∗k∗l∗t) = O(c).

3. The complexity for *matching* is O(n'*k*l*p*q) where k and l are the dimensions of the template and p and q are the dimension of the search window. For the reasons outlined in 2, this reduces to O(p*q).

The complexity of an iteration of the action-level servoing loop can now easily be determined by reviewing the principle steps in the loop.

Step	Complexity
1. Select n' landmarks from the locale containing the robot.	O(n∗m)
2. Take a picture	O(c)
3. For each landmark	
a) Construct a template for each of the n' vertices by ray tracing (template sizes are fixed).	O(c)
b) Match each template in the image using normalized correlation.	O(p∗q)
c) Compute heading error determine the correction.	O(c)

Total complexity of action level perceptual servoing	O(max(n∗m, p∗q)).

[6] Much of this discussion is summarized. A more detailed discussion can be found in [13] or [15].

The expression for total complexity is made up of two parts: a "reasoning" part of complexity $n*m$, and a "perception" part of complexity $p*q$. If we let s be the number of iterations of this loop for each unit of distance traveled, then for large s the motion increment between corrections will be small enough that robot can be modeled as traveling along a circular arc. Under these circumstances, the values of p and q need not be larger than: $p = a*(1/s)$ and $q = b*(1/s)$, where a and b are experimentally determined constants. Thus the complexity $\Pi_{REASONING}$ and $\Pi_{PERCEPTION}$ of an invocation of the reasoning and the perception parts of the loop can be expressed as:

$$\Pi_{PERCEPTION} = O\left(\frac{a}{s} * \frac{b}{s}\right) = O\left(\frac{1}{s^2}\right) \qquad \Pi_{REASONING} = O(n*m)$$

Similarly, the complexity of plan level perceptual servoing can be determined by analysis of the steps involved:

<u>Step</u>	<u>Complexity</u>
1. Select Landmarks (as above)	$O(n*m)$
2. Take a picture	$O(c)$
3. For each landmark	
a) Construct a template (as above)	$O(c)$
b) Match the template to image data (as above)	$O(p*q)$
c) Determine the pose from the matched data[7].	$O(n') = O(c)$
d) Adjust the plan sketch by replacing the start location for the next subgoal	$O(c)$
e) Refine the resulting subgoal, if it is not a primitive subgoal.	$\Pi_{PLANNING\text{-}INCREMENT}$

The action-level and plan-level servoing procedures use the same landmark selection technique. Moreover, the action-level servo loop is in continuous operation between milestone checks. Consequently, by the same argument used above, the p x q search window for matching will be just as it is in action-level servoing. Thus, the complexity of each invocation of reasoning and perception in this loop can be expressed as:

$$\Pi_{PERCEPTION} = O\left(\frac{1}{s^2}\right) \qquad \Pi_{REASONING} = O\left(\max\left(n*m, \Pi_{PLANNING-INCREMENT}\right)\right)$$

In both the action-level and plan-level servoing loops the cost of each iteration of perception decreases as $O(1/s^2)$. Consequently, the cost per unit of distance covered is $O(s*(1/s^2)) = O(1/s)$. The total cost of perception required to achieve a goal is $O(d/s)$ where d is the distance traveled. *This means that the total cost of perception required to achieve a goal decreases as the grain size of the RML system decreases.*

The n*m component of these expressions is a measure of the effort required to select landmarks. The RML architecture places a bound on n*m by limiting the scope of this reasoning to the locale in which motion is taking place (n and m are properties of this locale). Even so, it appears that the cost of this operation for a complete trip is O(d*s*n*m). For small s (infrequent use of vision) this would be true. The amount of change between images would be large enough to require repeating this reasoning for each s. However, as s increases the changes from image to image become increasing subtle. Consequently, for some s', repeating the reasoning more often than s' times per foot has no value. This means that the complexity expression can be tightened to O(d*s'*n*m) = O(d*n*m).

As a result of this analysis it can be seen that the total cost of achieving a goal can be expressed as

$$\Pi_{NAVIGATION} = O\left(d*\left(\frac{1}{s} + n*m\right) + \Pi_{TOTAL-PLANNING}\right)$$

where $\Pi_{TOTAL\text{-}PLANNING}$ is the total cost of planning the trip; this latter cost is the subject of the next section.

3.2. Planning complexity

To simplify the discussion, the analysis of planning complexity has been divided into three sections. The first of these, Section 3.2.1, begins by deriving an expression for the complexity of a single invocation of sketch-a-plan. This result is

[7] The algorithm used to determine pose in this system, pose-points [Kumar and Hanson [0], [0] 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 $i<5$.

used in Section 3.2.2 to determine the complexity of planning an entire route using sketch-a-plan and plan-and-monitor. These two sections show that, under the assumption that the environment is static and there are no errors in execution, the effect of the scope and perspective mechanisms is to induce a hierarchical plan development, resulting in a total planning effort of $O(\rho)$ where ρ is the number of segments in the final path. Section 3.2.3 analyzes the effects of execution errors and dynamic environments on planning complexity. It is argued that, under these fairly realistic circumstances, the fine grained RML paradigm reduces planning complexity to between $O(\rho \log(\rho))$ and $O(\rho)$ as compared to the $O(\rho^2)$ complexity of planning implemented as a macroprocess.

3.2.1. Complexity of plan sketching

One of the effects of representing an environment as a locale network is to set up a hierarchy of space partitions. This partitioning is a feature of the contained-by hierarchy described in Section 2.1. 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 contained-by hierarchy is β. When represented this way the total number η of nodes in the environment is:

$$\eta = \kappa*(1+\beta+\beta^2+\beta^3+\ldots+\beta^{\lambda-1}) = \kappa*\left(\frac{\beta^{\lambda}-1}{\beta-1}\right) \text{ where } \lambda = \log_\beta\left(\frac{\eta}{\kappa}\right)$$

The scope and perspective mechanisms dictate that the sketch-a-plan algorithm only consider nodes associated with the scope-locale and the offspring locales of that scope-locale. 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. Due to the η^{ρ} worst case behavior of A* this has the effect of limiting the complexity of a single invocation of sketch-a-plan to $O((\kappa*(1+\beta))^b)$ where b is the number of path segments returned.

3.2.2. 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 by recursively calling sketch-a-plan until all subgoals are primitive. It is straightforward to determine the complexity Π of generating plans this way. Letting b be the maximum number of subgoals constructed by sketch-a-plan, and ρ be total number of segments in the final path. The total number of times Γ that sketch-a-plan will be called is

$$\Gamma = O\left(1+b+b^2+\ldots+b^{\log_b(\rho)-1}\right) = O\left(\frac{b^{\log_b(\rho)}-1}{b-1}\right)$$

Since each call is $O((\kappa*(1+\beta))^b)$, the total complexity Π for generating the plan is

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

Since κ and β are bounded so is b, since no path would visit a node more than once. Hence, $(\kappa*(1+\beta))^b$ is bounded and so the total planning complexity is

$$\Pi = O(\rho).$$

The fact that RML planning linear is consistent with the results derived by Kleinrock and Kalmoun [22] and Korf [23] which show that it is theoretically possible to achieve linear planning complexity if planning is done hierarchically. Although RML plans are not developed in a strict hierarchical order, the complexity of producing a plan under the assumptions of this section is the same as if they were.

3.2.3. The effect of execution errors and dynamic environments on planning complexity

Changes in the environment and errors in 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. Moreover, 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. Based on experiments mentioned in Section 4.2.1 (Figure 24), however, 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 that .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.

In order to focus on the effects due to system architecture we can construct a macroprocess planner that can generate a detailed plan in O(ρ) operations. A very simple way to accomplish this is to modify the plan-and-monitor process so that it does not stop refining goals until *all* subgoals are primitive. Comparing this macroprocess planning algorithm to the RML planning algorithm in the presence of errors will show the advantage of the RML architecture when there are execution errors and environmental changes.

The macroprocess architecture 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. If it is necessary to replan n times the total planning effort for the macroprocess architecture can be expressed as:

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

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

The RML architecture develops a plan sketch in a depth first fashion until the first subgoal is primitive. Then action begins. Thus action takes place before the plan is fully developed. Ideally, this means that when an error or surprise is discovered, the planning process need only continue where it left off to plan the next step. This would keep the RML

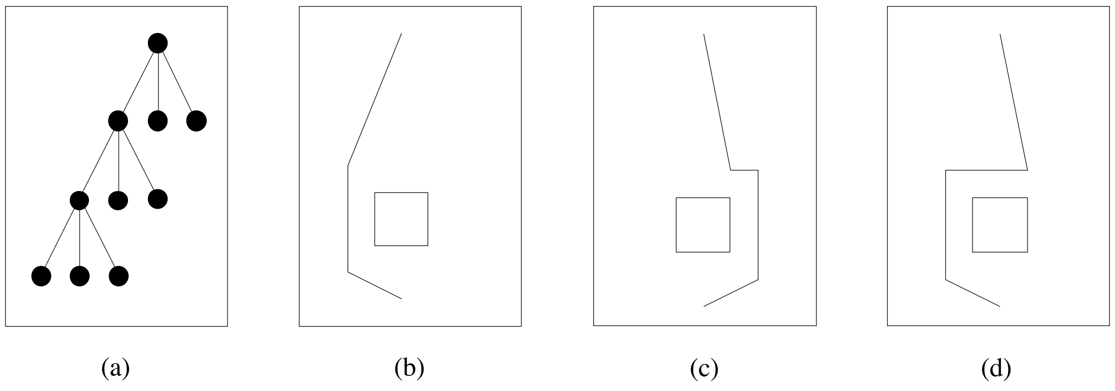


Figure 17 The RML architecture develops lazy plans, deferring most of the computation until it is required. Usually the first few subgoals of each plan sketch are primitive (a). Because of this, replanning is likely to be required after each detected error. Otherwise executing the first step of the ideal plan shown in (b) may not result in the smooth recovery of (c) but, rather the more erratic path shown in (d).

architecture planning complexity to the ideal O(ρ), 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 17 (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 17 (b), for example, may not result in the smooth recovery of Figure 17 (c), but rather the plan would attempt to continue by patching the old plan as in Figure 17 (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 kth replanning effort, the total planning effort for the RML architecture can be expressed as:

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

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, $\log_b(\pi_k) < (\pi_k - 1)$ for $1 < \pi_k$ $(= \rho_k)$. So $\Pi_{RML} < \Pi_{MAC}$. The RML architecture is more efficient in terms of its total planning complexity. The magnitude of the savings can be seen by observing that:

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

and

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

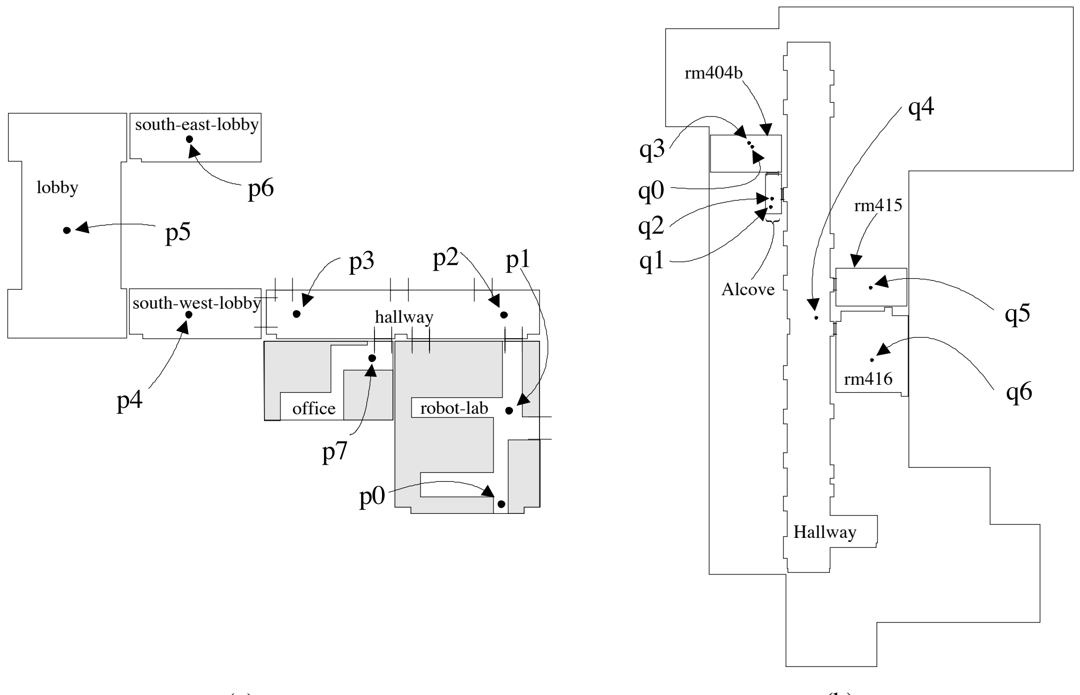


Figure 18 The planning experiments described in Section 4.1 were conducted using a model of the UMass Research Center (a) and a model of the Science Building at Mount Holyoke College (b). The goals identified in Figure 21 as (ptrans pj pk) and (ptrans qj qk) refer to the locations identified by the corresponding pj and qk, above. The gray areas in (a) correspond to obstacles in the office and robot lab.

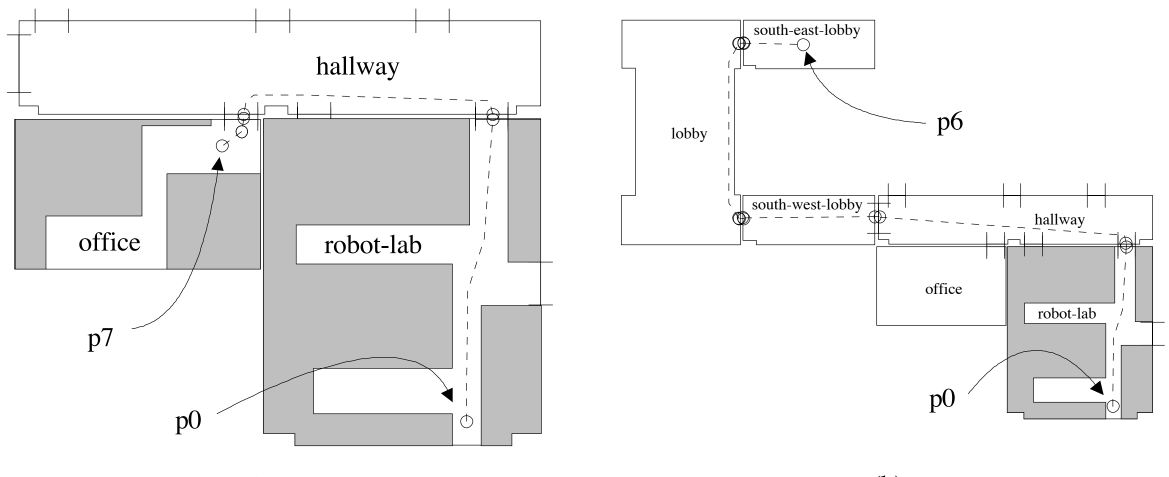


Figure 19 The path generated by plan-and-monitor in (a) is optimal. The path in (b) is not, but its length differs from the optimal path by less than 2%.

This savings is due to the fact that planning is done incrementally in the RML architecture.

A more intuitive expression for the total complexity of RML and macroprocess planning can be derived from these results if we make two assumptions. 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 Section 4.2. The second assumption is that the number of plan steps ρ_k generated by the kth 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 17 (c) and (d) show the results of replanning after an error in the execution of the plan shown in Figure 17 (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 17 (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

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

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

$$\Pi_{RML} = \left\{\log_b(\rho) + \log_b(\rho - 1) + \ldots + 1\right\} * \left(\kappa * (1 + \beta)\right)^b \le \rho * \log_b(\rho) * \left(\kappa * (1 + \beta)\right)^b = O\left(\rho * \log_b(\rho)\right).$$

These expressions indicate that the incremental nature of RML planning significantly reduces planning complexity when replanning is an issue.

In addition to the incremental nature of RML planning, the fine grain nature of the RML architecture means perception is used extensively to keep the vehicle executing actions accurately and according to plan. Since perception guided execution is capable of keeping errors quite small this increases the likelihood that plan sketches can used without being reconstructed. Execution of the plan of Figure 17 (b), for example, would result in only minor perturbations. This tends to bring the planning back to the incremental 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 incremental characteristic of RML planning 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.

4. Experimental Results

The analytical arguments of Section 3 support the claim that the RML architecture reduces the worst case complexity of the system. This section describes the experience gained from experiments that have been run at Mount Holyoke College and the University of Massachusetts. It is divided into two sections. Section 4.1 presents experiments with planning and Section 4.2 describes experiments with perception and perceptual servoing.

4.1 Planning

Experiments with planning have been conducted in two environments: the Lederlie Graduate Research Center at the University of Massachusetts (Figure 18(a)) and the Clapp Science Laboratory at Mount Holyoke College (Figure 18(b)). The data for these areas was collected by hand, using a steel tape to make careful measurements of the rooms, the objects, and the location of their visual features. These measurements, together with *rough* estimates of surface reflectances, were used to construct locale networks for each of the buildings.

The objective of these experiments was to explore the effectiveness of RML planning in “typical” situations. The three issues of particular importance were plan quality, complexity, and execution time. Plan quality was important because RML planning, as it has been implemented, is not guaranteed to produce optimal plans; experimental measurement of the complexity would test our analysis; and, since an objective for this system was real time performance, measurements of cpu times would help determine if this objective was realistic.

Several goals were designed to be "typical" of the kinds of goals a human would formulate when moving about the buildings and to result in a variety of path lengths. Each experiment consisted of presenting plan-and-monitor with a goal and recording the final path results, the complexity (in terms of the number of nodes expanded) and the cpu time.

4.1.1 Experiments, showing plan quality

The final path generated in these experiments show that, although final RML paths are not always optimal, they are quite "reasonable". The sample paths in Figure 19 and Figure 20 are typical. The departure from optimality in terms of distance traveled has been less than 2% in these experiments. Many paths have been optimal. When they are not, it has usually been due to the way doors are represented. For path generation purposes, doors are represented as a single point in the

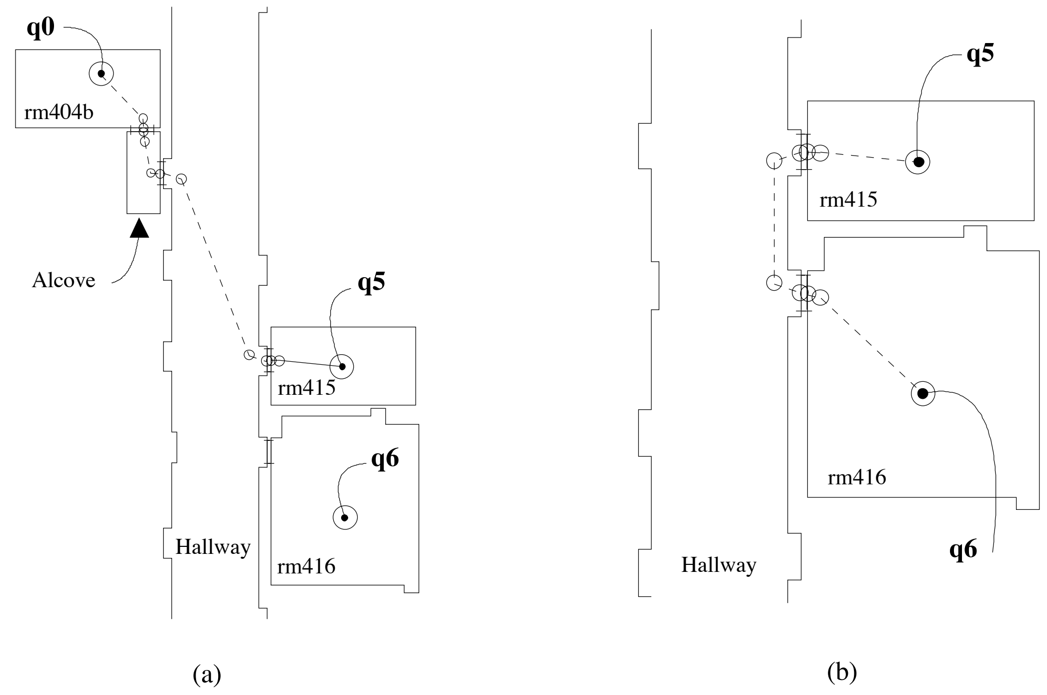


Figure 20 The plans generated by plan-and-monitor for (ptrans q0 q5), shown to in (a), and (ptrans q5 q6), shown in (b), are both optimal. These panels are details of the Mount Holyoke College environment shown in Figure 18b.

center of the intersection of the doorway on the floor (See Section 2.3.1). When a “door” is wide, as is the “door” between the lobby and the south-west lobby pictured in Figure 19 (b), the path will sometimes clear the edge of the door more than is necessary. The path in shown Figure 19 (b) is not optimal for this reason. This problem will be addressed as the system evolves.

It is also possible to formulate problems that will cause the current RML planner to produce sub-optimal plans by constructing a situation where the optimal path would require that the agent follow a path that leads outside the scope locale. One possible solution to this problem is to, once a path is found, increase the size of the scope locale so that it encloses all paths that are as long as the one that has been found. This could be done by selecting the smallest locale that encompasses the ellipse defined by the locus of locations p that satisfy the equation

distance(p, s) + distance(p, g) = d,

where s is the starting location, g is the goal location, and d is the length of the path that has been found. The optimal path would lie inside this new scope locale, since any path that would leave that new locale would necessarily be longer than the one that has already been found.

There are other factors that put optimality at risk. The details of the RML planner will change as the system evolves. The current system does, however, generate quality plans in the environments in which it has been tested.

Goal Label	ρM	CPU Time	Nodes Expanded	Calls To Reachable	Goal Description
goal-1	3	3 seconds	6	89	(ptrans p0 p1)
goal-2	8	6 seconds	18	168	(ptrans p0 p2)
goal-3	8	7 seconds	18	168	(ptrans p0 p3)
goal-4	11	8 seconds	25	169	(ptrans p0 p4)
goal-5	16	10 seconds	41	223	(ptrans p0 p5)
goal-6	19	17 seconds	55	320	(ptrans p0 p6)
goal-7	15	11 seconds	43	252	(ptrans p0 p7)
goal-8	5	9 seconds	6	8	(ptrans q0 q1)
goal-9	1	1 second	1	2	(ptrans q0 q3)
goal-10	11	53 seconds	61	251	(ptrans q4 q3)
goal-11	16	35 seconds	65	173	(ptrans q0 q5)
goal-12	16	61 seconds	78	312	(ptrans q0 q6)
goal-13	11	38 seconds	57	159	(ptrans q5 q6)

(a)

Goal Label	ρA	CPU Time	Nodes Expanded	Calls To Reachable	Goal Description
goal-1	3	3 seconds	13	146	(ptrans p0 p1)
goal-2	8	14 seconds	77	626	(ptrans p0 p2)

(b)

Goal Label	ρA	CPU Time	Nodes Expanded	Calls To Reachable	Goal Description
goal-1	3	90 seconds	13	785	(ptrans p0 p1)
goal-2	8	1860 seconds	267	10181	(ptrans p0 p2)

(c)

Figure 21 The results shown in (a) summarize planning complexity experience when the plan-and-monitor algorithm was used. These data were collected from experiments run in the University of Massachusetts and Mount Holyoke campuses. For comparison purposes, the results in (b) and (c) show the effect of applying A* (unfocused) to the same goals used in (a). In (b) the model used consisted only of the robot-lab shown in Figure 19 (a). Expanding the model to include the entire second-floor of the research building increased the measured complexity for the A* algorithm considerably (b). These experiments were performed on a 33 megahertz Intel 80486 based machine with 10 megabytes of RAM. All code was written in POPLOG COMMONLISP.

4.1.2. Experimental Experience with Planning Complexity in Typical Situations.

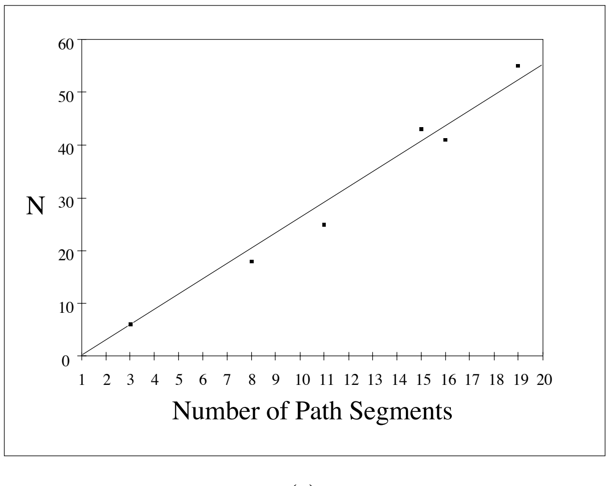
The results of complexity measurements for Plan and Monitor are shown in Figure 21. Each row in the figure, shows the results of an experiment with a goal constructed from a pair of points from the set {p0, ..., p7, q0, ..., q6} identified in Figure 18. Figure 21 (a) illustrates the performance of plan and monitor. For comparison purposes, Figure 21 (b) and (c) present results of experiments with the unfocused use of A*.

The first thing to notice is that the experiments labeled goal-1, ..., goal-7 of Figure 21 (a) are consistent with result that the complexity of plan-and-monitor is O(ρ). The results shown in Figure 21 (b) and (c) are consistent with the exponential growth predictions for A*. This is made more by the graphs presented in Figure 22. The experiments with A* were run with two different models to show the effects of model size. One set of experiments was run using the room labeled "robot-lab" in Figure 18(a) as the environment. These results are illustrated in Figure 21 (b). For this case, only goal-1 and goal-2 are relevant (all others are outside the robot-lab environment). The goal-1 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 used to determine 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 planning points outside the robot-lab from consideration. The unfocused A* does not.

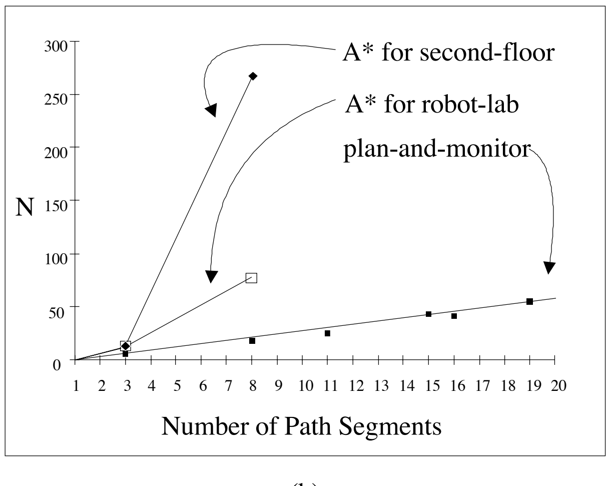
A second set of A* experiments were run on the entire environment shown in Figure 18 (a). The results for goal-1 begin to show a significant increase in cpu time. This is clearly due to the increase in the number of calls to reachable. The number of nodes expanded remains the same as for Figure 21 (b), not remarkably different from plan-and-monitor. The result for goal-2, however, shows considerable growth in the number of nodes expanded when compared with plan-and-monitor. 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 data in Figure 21 (b) and (c) shows the effect of increasing the size of the model (the number of nodes) on A*. This consistent with the predicted $O(\eta^{\rho})$ growth characteristics of A*. 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 21 (c). It is not, however, a surprise complexity issue. This is because the successor function is called for each node expanded. This successor function, in turn, 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. For comparison purposes, the A* results are combined with those for plan-and-monitor in Figure 22 (b).

A second observation is that, although the behavior of plan-and-monitor on the goals goal-8, ..., goal-13 is significantly better than the results reported for A*, it does not exhibit the approximately linear results that goal-1, ... , goal-7 exhibit in

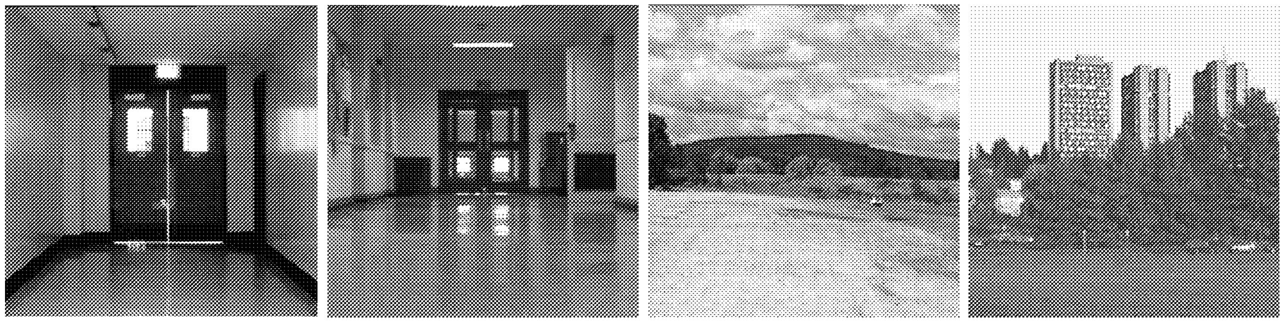


(a)



(b)

Figure 22 The number of nodes expanded as a function of path length is shown for "plan-and-monitor" in (a). The straight line, drawn for reference, shows the complexity to be approximately linear for path lengths up to length 20, the longest path attempted. Plan-and-monitor and the unfocused use of A* are shown together in (b).



(a) (b) (c) (d)

Figure 23 Perceptual servoing experiments have been conducted successfully under a broad variety of conditions. The initial experiments were conducted in 1989 with Harvey [11] (a). Similar experiments have been conducted since with Susan B., but these were conducted under a broad variety of lighting conditions [14][7]. The most recent experiments were conducted in 1993 in an outdoor environment using the HUMMWV (c) and (d).

Figure 22 (a). This, too, is consistent with the analytical results. Recall that complexity of RML planning is

$$\Pi_{\text{RML}} = O\left(\left(\frac{b^{\log_b(\rho)} - 1}{b - 1}\right) * \left(\kappa * (1 + \beta)\right)^b\right) = O\left((\rho - 1) * \left(\kappa * (1 + \beta)\right)^b\right)$$

where ρ is the total number of segments in the final path, κ is the number of planning nodes associated with each locale, β is the branching factor of the contained-by hierarchy, and b is the number of subgoals returned by sketch-a-plan. The goals goal-1, ... , goal-7 refer to environment shown in Figure 18 (a). In this environment κ is small and approximately the same for each locale. This environment is a "textbook case". The second set of goals, goal-8, ..., goal-13, are from Figure 18 (b). Most of the locales in this environment also have a small κ. The hallway locale, however, is an exception. It is very complex and has a very large κ associated with it. The data reflects the complexity of planning in this locale. This situation can be remedied by subdividing the hallway into sublocales.

A third observation is that the cpu times for plan-and-monitor comfortably small. For goal-1, ... , goal-7 the times are similar to the times it takes a human to walk the paths that are generated.

4.2 Perception

Experiments with RML perception have been conducted under a variety of conditions in both indoor and outdoor environments. Some experiments have been conducted on individual images to test how well the matching technique behaves under variations in ambient lighting [14], but most experiments have been run using perception to control a vehicle. Figure 23 contains images that were part of the sequence of images used in these experiments. Figure 23 (a) is typical of the images acquired during the action level and plan level servoing experiments done with “Harvey” as reported in [13]. Figure 23 (b) is typical of images acquired during experiments conducted with “Susan B.” in the hallways of Mount Holyoke Campus [7], [14], [21], and [26]. The most recent experiments have been conducted outdoors with the HUMMWV. Figure 23 (c) and Figure 23(d) are images from those experiments . Indoor experiments have been conducted successfully in a broad range of ambient lighting. The outdoor experiments were successfully performed in a variety of weather conditions, including light rain, fog, clear and sunny, and cloudy. The matching technique used by action-level and plan-level servoing seems very robust.

The remainder of this section shows the results of experiments with two of the perceptual servoing loops. Section 4.2.1 presents data relating to action level servoing and Section 4.2.3 discusses experiments with plan level servoing.

4.2.1 Action Level Servoing

Experiments described in Fennema [13] show that even a well built robot like “Harvey” will deviate considerably from its course when no sensory feedback is used. When action-level servoing is used the robot's trajectory has been controlled to within .3" of its intended course for distances as long as 40 feet (Figure 24).

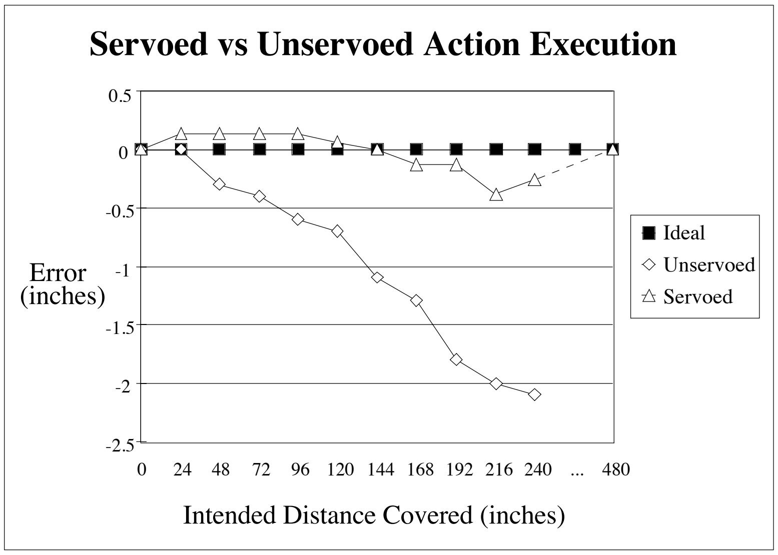


Figure 24 Using action-level perceptual servoing dramatically improves the ability of the robot to remain on course. When attempting to traverse a straight line path the servoed robot (△) deviates no more than .3 inches from the straight line for as much as 40 ft (480 inches). Without servoing (◇) the robot soon wanders from the intended path.

Action-level servoing experiments have been conducted by giving the agent a command to move along a straight path and measuring how far the agent would deviate from that line. Experiments with "Harvey" produced the results shown in Figure 24. Figure 23(a) is an image acquired during these tests. Experiments with the "Susan B." platform were more challenging. "Susan B." has very poor direction control and "her" speed can vary considerably during the course of an experiment. In an unservoed mode "she" typically wanders off course as much as 2 inches per foot of travel. Experiments with a modified action level servoing loop have been performed at 15 iterations per second has kept "Susan B." within 2 inches of "her" intended course at speeds of 2 feet per second for 20 feet [7][8]. Figure 23 (b) is typical of the images acquired during those experiments. The most recent experiments in action level servoing have been performed outdoors with the HUMMWV. During these experiments the vehicle was kept within 6 inches of its intended path for 450 feet at a speed of 3 feet per second. Figure 23 (c) and (d) were acquired during those experiments. Moving perceptual servoing to outdoor environments was surprisingly easy. The images in Figure 23 reveal one of the reasons. Few of the surfaces in outdoor scenes such as Figure 23 (c) show any specularity and none of the specular surfaces (windows) in Figure 23 (d) were used for landmarks.

On the basis of these results, it seems reasonable to believe that primitive actions can be executed accurately enough to offer significant control over the amount of replanning necessary. This makes a strong case for the validity of the assumption put forth in support of the arguments made in section 3.2.3. It seems that action-level servoing can and does work, at least *in the sense that it can be demonstrated.* Showing that it can be made to work reliably over a broad range of conditions is a goal of this investigation.

4.2.2. Plan-Level-Perceptual Servoing

As mentioned in Section 2.4.5, plan-level servoing uses the same landmark selection, template construction and matching operations that action-level servoing uses. The only difference is that it uses this information as input to the pose determination algorithm described by Kumar and Hanson[24][25].

Most experiments that relate to plan level servoing have been tests to determine the accuracy of the pose refinement algorithm. Typical experiments use a single image, similar to the one shown in Figure 23 (a). Each test would begin by entering a robot location into the system. This became the location where the agent expected to be. This "expected" location was used by landmark selection, template construction and matching operations. The results of the matching would be input to the pose determination program. The result, the "measured" pose, was then compared with the actual pose.

	Left 3 inches	Actual	Right 3 inches
Expected			
x	40.00	40.00	40.00
y	3.75	4.00	4.25
z	0.00	0.00	0.00
theta-x	0.00	0.00	0.00
theta-y	0.00	0.00	0.00
theta-z	3.14	3.14	3.14
Measured			
x	39.93	40.11	39.85
y	4.01	4.08	4.17
z	-0.22	0.20	0.00
theta-x	-0.0047	-0.0090	-0.0050
theta-y	-0.0151	-0.0050	-0.0093
theta-z	3.14	3.14	3.14

Figure 25 3D pose determination experiments have shown the ability to find the robot's position to within 1.5 inches. This table shows the results of determining the robot's pose using perception. The center column shows the pose measurement determined when the robot's expectations were correct. The other two columns show results when the robot erroneously expected to be to the left or right of its actual position. In all cases was the actual robot pose in the locale was x=40 feet, y=4 feet, z=0 feet, theta-x=0, theta-y=0 and theta-z=3.1415 radians.

Figure 25 shows the results of a typical experiment. In the column marked "Left 3 inches" the expected location is 3 inches to the left of the actual location. The measured pose in this case was (39.93, 4.01, -0.22). The actual pose was (40, 4, 0). The error was less than .5 inches. The other results shown in this figure are also very accurate, as are the results reported by Kumar and Hanson. Since action level inaccuracies are on the order of inches, this level of accuracy is high enough to provide a useful milestone recognition capability. A goal of this work is to show that it can be made to work reliably over a broad range of conditions.

The entire RML system, including action level and plan level servoing, 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.

5. Conclusions and Future Work

This initial investigation of the RML style interweaving of reason, perception and action has presented evidence that moving from a coarse-grained macroprocess architecture to a fine-grained interweaving of these processes significantly improves total system complexity. The complexity analysis in Section 3 showed that incremental, fine-grained RML interaction improves reasoning (planning) complexity in real situations, when errors and surprises can occur. It was shown that, under these circumstances, the planning complexity of a macroprocess system would be $O(\rho^2)$, whereas the RML system would be between $O(\rho*\log(\rho))$ and $O(\rho)$, where ρ is the number of segments in the final path. The complexity of perception (vision) in the RML system was shown to be $O(d/s)$, where s is the number of times vision is invoked per unit of distance traveled, and d is the total length of the trip.

The experimental results described in Section 4 provide evidence that the RML concept can work in practice and that the savings predicted by the analysis can be realized. Experiments with RML perceptual servoing show that it can be performed in real time, that it works in a variety of environments, and that it can produce accurate motion. The resulting motion accuracy is clearly good enough to satisfy the requirements to keep planning complexity to $O(\rho)$.

These results make a statement, but it must be remembered that the evidence presented is for the subsystem shaded in Figure 2. A number of questions remain.

1. Can the other capabilities identified in Figure 2 be implemented as RML processes? How will their addition of affect the complexity of the system? Can it be shown that interweaving continues to produce quantifiable efficiencies?

2. There are numerous questions about the system as it is described in this paper. Can the representations be based on something other than polyhedra? How does this affect the reasoning, perception processes and their complexities? Can the perspective and scope mechanisms be implemented or used differently? How does this affect complexity and plan quality?

3. How reliable are action-level and plan-level servoing? How tolerant is the system to representational inaccuracies?

4. How can the concepts described in this research be formalized so they can be used by others, short of reproducing the system.

Work in progress addresses questions 1 and 2. Research at Mount Holyoke College, for example is currently being conducted in the use of vision for model acquisition, for determining robot location, for obstacle avoidance, and for more robust, real time action-level servoing. Lakshmi Ratan [26] has been successful at adding regions and reflectance values to faces in an existing locale network. This adds the information used by action-level and plan-level servoing. Another recent project used an RML loop to determine which locale contains the robot by matching modeled floorplans to visual data by Kapur [21]. Finally, Chuah and Fennema [7],[8] use vision to continually estimate the robots position as part of an action-level servoing loop.

Reliability and tolerance questions are often referred to as "engineering" tasks. Perhaps this is because answering them from a scientific perspective is so difficult. The answers tend to be highly domain specific. As with other systems that claim to be "robust", the RML concepts have been *demonstrated* in a variety of settings with different mobile robots. This does not, however, predict what will happen in a new situation. It is a matter of considerable importance to the AI community to find ways to express this dimension of our work.

Finally, there is the need to formalize and abstract. The primary point of this paper is that the RML style interweaving of reason, perception and action is efficient. This is an important and useful point. Another is that organizing space into a contained-by hierarchy provides a powerful way of focusing processes that reason about planning and perception. As this research project progresses, uncovering and strengthening these abstractions will be of primary importance.

References

1 Aloimonos, Yiannis. "Behavioral Visual Motion Analysis". Proceedings of the 1992 DARPA Image Understanding Workshop, pp. 521-541, San Mateo, CA: Morgan Koffman Publishers, 1992.

2 Arkin, Ronald C. "Towards Cosmopolitan Robots: Intelligent Navigation in Extended Man-Made Environments." Ph.D. Dissertation, University of Massachusetts, September 1987.

3 Bares, J., Herbert, M., Kanade, T., Krotkov, E., Mitchell, T., Simmons, R., and Wittaker, W. "Ambler: An Autonomouse Rover for Planetary Exploration." IEEE Computer Magazine, pp. 18-26, June 1989.

4 Bajcsy, Ruzena. "An Active Observer". Proceedings of the 1992 DARPA Image Understanding Workshop, pp. 137-147, San Mateo, CA: Morgan Koffman Publishers, 1992.

5 Brooks, Frederick P., "The Mythical Man-Month", Addison Wesley (1978)

6 Brooks, Rodney A. "A Robust Layered Control System For a Mobile Robot". IEEE Journal of Robotics and Automation, Vol. RA-2, No. 1., pp. 14-23, March 1986.

7 Chuah, Mei Ching. "Steering a Robot in Real Time with a Parallel System." ." Honors Thesis, Mount Holyoke College, South Hadley, MA, 1993.

8 Chuah, Mei Ching and Fennema, Claude L. "Steering a Mobile Robot in Real Time." To be published in Proceedings of SPIE Intelligent Robots and Computer Vision XIII, Boston MA, September 1994.

9 Elfes, Alberto. "A distributed control architecture for an autonomous mobile robot." Artificial Intelligence, Vol. 1, No. 2, 1986.

10 Fennema, Claude L. Jr., Riseman, Edward M. and Hanson, Allen R., "Planning With Perceptual Milestones to Control Uncertainty in Robot Navigation", Proceedings of SPIE Mobile Robots III, pp. 2-18, November 1988.

11 Fennema, Claude L. and Hanson, Allen R. "Experiments in Autonomous Navigation." Proceedings of the Tenth International Conference on Pattern Recognition, Atlantic City, New Jersey, June 16-21, 1990, pp. 24-31, Los Alamitos, CA: IEEE Computer Society, 1990.

12 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.

13 Fennema, Claude L. "Interweaving Reason, Action and Perception." PhD. Dissertation, University of Massachusetts, Amherst, September 1991.

14 Fennema, Claude L. “Finding Landmark Features Under a Broad Range of Lighting Conditions.” Proceedings of the SPIE Intelligent Robots and Computer Vision XII: Algorithms and Techniques, Vol 2055, pp. 181-191, Boston MA, September 1993.

15 Fennema, Claude L. “Interweaving Reason, Action and Perception” Technical Report, COINS Department, University of Massachusetts, Amherst MA, 1994.

16 Gat, Erann, Slack, Marc, G., Miller, David P. and Firby, R. James. “Path Planning and Execution Monitoring for a Planetary Rover.” Proceedings of the IEEE International Conference on Robotics and Automation, pp. 20-25, 1990.

17 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), pp. 100-107, 1968.

18 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.

19 Hendler, James, Tate, Austin and Drummond, Mark. "AI Planning: Systems and Techniques". AI Magazine, pp. 61-77, Summer 1990.

20 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.

21 Kapur, Tina. "Where Am I? - Using Vision to Localize a Robot." Honors Thesis, Mount Holyoke College, South Hadley, MA, 1992.

22 Kleinrock, L. and Kamoun, F. "Hierarchical routing for large networks". Comp. Networks 1, pp. 155-174, 1977.

23 Korf, R. E. "Planning as Search: A Quantitative Approach." Artificial Intelligence 33:65-68, 1987.

24 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.

25 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.

26 Lakshmi Ratan, Aparna. "Acquiring new knowledge about the environment using computer vision." Honors Thesis, Mount Holyoke College, South Hadley, MA, 1992.

27 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.

28 Minsky, Marvin. "The society of mind". New York: Simon and Schuster, 1986.

29 Moravec, Hans P. "The Stanford Cart and the CMU Rover". IEEE Proceedings, Vol. 71, No. 7, July 1983.

30 Nilsson, Nils J. "Shakey the Robot". SRI Technical Note 323, 1984.

31 Panton, D., Rosenblatt, J. K., and Keirsey, D. M. “Plan Guided Reaction.” IEEE Transactions on Systems, Man and Cybernetics, Vol. 20, No. 6, pp. 1370-1382, November/December 1990.

32 Pomerleau, Dean A. "Neural Network Based Autonomous Navigation". Appeared in "Vision and Navigation", Charles Thorpe (ed.), Kluwer Academic Publishers, 1990.

33 Raphael, Bertram. "The Thinking Computer: Mind Inside Matter." W. H. Freeman and Company, New York, 1976.

34 Schank, Roger and Abelson, Robert. "Scripts Plans Goals and Understanding". Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1976.

35 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.

36 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.

37 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.