RESOURCE-COORDINATED HIERARCHICAL PLANNING FOR REAL-TIME AUTONOMOUS SYSTEMS

by

William D. Hall

S. B. A. A., Massachusetts Institute of Technology

(1991)

SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN
AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1992

Signature of Author _____
Department of Aeronautics and Astronautics
June, 1992

Certified by _____
Dr. Milton B. Adams
Lecturer, Thesis Supervisor and CSDL Technical Supervisor

Accepted by _____
Professor Harold Y. Wachman
Chairman, Departmental Graduate Committee

Resource-Coordinated Hierarchical Planning for Real-Time Autonomous Systems

by

William D. Hall

Submitted to the Department of Aeronautics and Astronautics
on May 8, 1992 in partial fulfillment of the requirements for the
Degree of Master of Science in Aeronautics and Astronautics

ABSTRACT

In recent years, mobile robots have been used to perform tasks unsuitable for humans, be they too dangerous or logistically difficult. Their use has allowed undertakings which would have otherwise been impossible or economically infeasible. Still, robot technology is quite limited. In particular, many of the more challenging robotic tasks require a person to actively control the robot and make the decisions necessary for completing the task. In a number of applications, the requirement for a person to be in control of the robot is prohibitive; these applications require rapid, flawless communication between the robot and the operator. Removal of the communications link would allow whole new fields of research and enterprise, particularly in underwater and space applications, where communication is technically difficult and of low quality.

The capabilities required of an autonomous mobile robotic system include the ability to make and to execute plans. In order to execute plans, the system must make the decisions necessary to execute those plans, and it must determine when replanning is necessary. The planning problem is one that is often considered in the context of optimization theory. The solutions to the problem of planning in a *real-time* scenario and to the problem of executing those plans require further development.

In this thesis, the theory underlying the real-time planning problem is discussed and developed in the context of a hierarchical optimization problem. A focus of this thesis research has been the development of the management functions that are required for the real-time coordination of the planning and plan execution performed at each level of a hierarchical planning system. A software architecture has been designed and implemented; the implementation includes the hierarchy of planners and the associated planning management functions. The architecture is applied in an autonomous underwater vehicle scenario. The resulting planning system is tested in a simulation wherein commands are issued to a control system which in turn drives a six-degree-of-freedom model of a small submersible. Models of sensors provide feedback information to the planning system, closing the loop.

The main result of the research has been the formulation of an autonomous planning system architecture which is applicable to a variety of planning and decision-making problems. Further development of the algorithms employed in this embodiment of the planning system can improve the performance of the system when applied to any particular problem. It is the development of an architecture that integrates and manages a *real-time* hierarchical planning system, not the particular algorithms used in its embodiment, that is the major contribution of the thesis.

Thesis Supervisor: Milton B. Adams
Title: Lecturer in Aeronautics and Astronautics

ACKNOWLEDGEMENT

The author would like to express his gratitude to Milt Adams for the invaluable aid he provided in the definition and preparation of this thesis, to Bob Powers for his help in the initial phases of research, to Doug Humphrey, Noel Nistler, and Steve Atkins for their good cheer and assistance with computer-related problems, and especially to Melitta King for the encouragement and support she supplied throughout the project.

This thesis was prepared at The Charles Stark Draper Laboratory, Inc., under Internal Research and Development Contract #18634. Publication of this report does not constitute approval by the Draper Laboratory of the findings or conclusions contained herein. It is published for the exchange and stimulation of ideas.

1. INTRODUCTION

The future holds great promise for autonomous mobile robot systems. They don't require the life-support systems required on manned systems, and they don't risk human life in dangerous situations. In the immediate future, they will be applied to tasks in which logistics preclude a human operator. The spaceships and planetary rovers for a mission to Mars, the submersibles needed for a scientific effort to monitor areas of the sea floor over a long period of time, and the aircraft used for wartime surveillance under battlefield conditions are all examples of systems which can benefit substantially from autonomy. Systems such as these will require solutions to many of the basic problems of autonomy.

Autonomous systems will become economically attractive as their capabilities develop, since less research will be required to develop any particular system than is currently required. Their broader range of capabilities will support a much wider spectrum of applications. Applications in which predictable, repeatable performance is required and applications which incur great costs in training human operators are clear candidates for employment of autonomous systems. As autonomous systems become more reliable and cost effective, they will command a larger and larger market.

1.1. PROBLEM STATEMENT

Currently, the development of an autonomous system requires extensive resources of money and development time. Every autonomous system must be carefully designed to be reliable and predictable in performing its tasks; the designers are required to consider every detail of the system from its control systems to its failure modes, as well as the physical system's basic design.

The design of onboard software is complicated, involved, and perhaps the most critical element to the success of the system. The onboard software must be able to conduct the mission, manage the system's resources, and ensure that the system's resource constraints are met. It must also manage the computational resources it uses to ensure an adequate real-time response, and it must interact with the operating system. However, all autonomous systems share many of these same functional requirements, each requiring their application in a specialized way which is capable of solving the problem for that particular system. What is needed is a general software architecture which accommodates the details of planning, resource management, modeling, and so forth, and that is versatile enough to be applied to a whole class of autonomous mobile robot problems. Such an architecture would drastically reduce the development time and cost associated with autonomous systems

1.2. OBJECTIVES

The objectives of this research are to determine those functional requirements that are common to most autonomous systems and to develop a framework within which these functions can be performed for any particular system. The customization required for a particular system ideally should require only that the designers provide that framework with models of the system.

The use of this framework is demonstrated in this thesis in the context of an autonomous underwater vehicle application. The capabilities, models, and tasks of the autonomous vehicle are specified in terms of a set of goals, the consumable resources of the system, and functions which estimate the amount of those resources used during execution of each of the system's goals. These resource-use estimation functions contain the vehicle-specific knowledge necessary to plan a mission, and the goal definitions contain the information required by the system to convert the plans into actions.

The framework is evaluated in a simulation environment. The effects of the architecture's coordination parameters on the vehicle's real-time performance in various scenarios are examined through simulation experiments. The scenarios vary in the degree of uncertainty of the information available to the system, the reliability of the information, and the timeliness of the information.

The objective of this research is not to develop the ultimate set of algorithms to perform the architecture's management and planning functions, nor is it to find the optimum parameters for the algorithms that are used. Rather, this thesis seeks to demonstrate the feasibility of the concept of a general real-time onboard architecture, and to determine an effective structure for such an architecture. The form of this structure is determined by the functional relationships between the elements of the architecture (these elements are described in Chapter 4), namely, the inputs and outputs of the algorithms employed in the architecture and not the specific algorithms themselves. Any shortcomings in the particular planning, management, and coordination algorithms implemented in the embodiment of the architecture employed for this thesis should not be mistaken for fundamental flaws of the architecture's structure. The simulations are meant to show the feasibility of the architecture's structure. Indeed, the overall effectiveness can be improved for any particular application with further development of the individual algorithms.

1.3. OVERVIEW

Chapter 2 presents a brief overview of a variety of approaches to autonomous underwater vehicle architectures that have been described in the literature. Chapter 3 analyzes these approaches in the context of large scale control and decision system techniques and determines which elements the architectures hold in common. Chapter 4 proposes the features of the general architectural framework discussed above and

describes the implementation of that framework. Chapter 5 explains the control mechanisms used by the architecture to manage and coordinate its disparate functions. Chapter 6 details the implementation of the entire system with the focus on the management and coordination control mechanisms. Chapter 7 describes the tests and associated scenarios employed to evaluate the management control mechanisms and discusses the results of those tests. Chapter 8 concludes the thesis and proposes areas for further research.

2. BACKGROUND

Experience with autonomous underwater vehicle (AUV) software architectures is broadening as the research community continues to deploy and test autonomous vehicles. Here *software architecture* refers to the implementation of the onboard decision-making, planning and sensor processing functions and the mechanisms by which these functions interact to provide the onboard intelligence required by an underwater vehicle to autonomously perform its mission. A variety of designs for AUV software architectures have been implemented, and many more have been proposed, but little has been done to assimilate and review this rapidly expanding body of experience. The analysis of this collection of experience that is presented in this section points to some significant issues which affect the performance of any vehicle; a better understanding of these issues should help AUV designers create more reliable and versatile vehicles. The development of AUV's serves as one of the leading areas of planning, decision-making, and control research for autonomous mobile robots. Thus, the overview of the current state of the art in AUV software architectures presented here yields insight into general autonomous system issues.

The twelve vehicle architectures reviewed here are those developed for: the Naval Postgraduate School's vehicle [1]; Honeywell's vehicle [7]; the tethered autonomous vehicle at Heriot-Watt University [10]; International Submarine Engineering's work [13]; SINTEF's vehicle [5], [6]; a group of vehicles based on Rodney Brooks' layered architecture [2] including the Draper-MIT Sea Squirt [3], [4]; Martin Marietta's vehicle [11]; the work done at Hughes [8], [9]; Carnegie-Mellon's Ambler [14]; the National Institute of Standards and Technology's reference architecture [16]; the work of LAAS [12]; and the architecture designed at Linköping University [15].

2.1. ONBOARD SOFTWARE ARCHITECTURES

A brief overview of the characteristics of the onboard software architectures found in existing autonomous vehicle control schemes is presented here. Five aspects of each architecture are examined:

(1) the organization of the architecture into a hierarchy or heterarchy,

(2) the use of information abstractions in the architecture,

(3) the use of modeling and models,

(4) the type of knowledge that is made available to different elements of the architecture, and

(5) the type of knowledge maintained within the architecture.

These characteristics differentiate architectures according to their capabilities as described below.

2.1.1. Hierarchical versus Heterarchical

The plan-following ability of a system is influenced by its architectural organization as either a hierarchy or a heterarchy. A hierarchical architecture is arranged in a stratum of levels, with the top levels commanding bottom levels [19]. Generally, the top level of a hierarchy is concerned with the goals of the overall mission, and it directs the lower levels to solve the individual detailed problems upon which the goals at the top level are dependent. A heterarchy, on the other hand, is organized into separate architectural units, sometimes called behaviors, each of which vies for control of the autonomous system through a command arbitration protocol [2]. The command arbitration protocol is responsible for ensuring that the most appropriate behavior for the mission situation as assessed onboard maintains control of the vehicle. Often, the behaviors that are denied control of the vehicle either ignore the fact that they have been denied control or they must determine that fact for themselves.

A hierarchy is naturally suited to following plans whose original description is

given at only an abstract level of detail. The details of the plan are worked out by the interactions among the levels of the hierarchy as the plan is executed. Tasks are often classified into levels of abstraction in order to more easily understand them, so it is natural for a person to arrange the goals of a system hierarchically, at different levels of abstraction, when defining that system. In addition, many problems tend to break down naturally into hierarchical levels, and their solutions are simplified if expressed as such [19].

Heterarchies, on the other hand, are typically employed to control robots to exhibit desired behaviors rather than to coordinate problem solving or plan following [2]. They can react very quickly to newly assessed situations, since they only need to switch the architectural unit that controls the vehicle's behavior to respond to the new situation. The drawback of a heterarchy is that a separate behavior must be developed and coded for each expected class of situation, and an arbitration protocol must be designed to recognize each situation so that it can pass control to the appropriate behavior. The difficulty in creating an effective yet simple arbitration protocol is that the arbitration scheme does not have access to the knowledge contained in the behaviors, so it must arbitrate among things about which it has limited information.

2.1.1.1. Brooks' Layered Control: The Classical Heterarchy

Rodney Brooks first documented the use of the layered control approach in an application to land robots [2]. His version of layered control is heterarchical, and each module of the heterarchy acquires information at the required level of abstraction (i.e., level of detail) solely from basic sensor data. The modules of the heterarchy are responsible for performing tasks at different levels of abstraction; the modules which perform the tasks at the lowest levels of abstraction (i.e., highest level of detail) have the greatest priority in the command arbitration protocol. For example, a module responsible for stopping the robot if it is about to run into an obstacle is at a low level of abstraction and would have priority over a module responsible for recognizing obstacles at a distance

and planning a path around them. Brooks' research is directed towards making robots appear and behave life-like, rather than on having them follow instructions, and for those purposes the pure form of his architecture works well.

Others have applied the layered control architecture to controlling AUV's, notably the joint venture between MIT Sea Grant and C.S. Draper Laboratory [3]. With the lack of control over coordination in such systems, and the fact that the modules of the architectures are rooted in the details of the particular vehicle's design, they are not well suited to planning or to following instructions. This poor plan-following ability has motivated researchers to add hierarchical elements to the MIT/ Draper system [4].

2.1.1.2. The NIST Reference Architecture: A Hierarchical Example

James Albus has proposed a hierarchical architecture for the control of autonomous vehicles [16]. It is organized so that the complexity, abstraction, and time scale of the plans and information created and used at each level are consistent: abstract, long-term information which has been gathered from many sources is used at the higher levels, and is updated from the detailed yet simple, short-term information acquired at the lower levels. Sensory processing, world modeling, and task decomposition occur at each of the layers. The sensory processing and modeling functions build up the information into complex and abstract forms in the higher levels. Task decomposition functions break down the vehicle's abstract, high-level goals into specific, detailed commands at low levels which ultimately result in commanded vehicle actions.

The reactivity of the system is supported by the temporal decomposition of the hierarchy. Quick, nearly reflexive decisions can be made by the lowest level since it need only consider its fairly simple goals and information at a low level of abstraction. Decisions which require more planning time and effort because they affect a greater portion of the mission are made at a higher level whose scope is commensurate with that required of the situation at hand.

2.1.2. Structural Decomposition

Some architectures are broken down according to specific vehicle activities, having one architectural unit execute all phases of an activity such as a transit or survey. Others are broken down into abstract functions such as planning, modeling, and vehicle control all of which contribute to the execution of any given vehicle activity. The former approach is more amenable to the performance of specific, desired tasks, since they are directly programmed into the vehicle's behavior. Adding to or modifying the repertoire of tasks can be difficult, however, since each adjustment may require a modification of many of the architecture's parts. The interfaces between parts of the architecture are based on the particular vehicle activities served by those architectural parts. Since the vehicle activities are very interdependent, changing or adding to the vehicle's capabilities may require redefining interfaces throughout the system.

The latter approach, namely, using one part of the architecture to perform abstract functions common to all vehicle tasks, is more general but requires that more work be done to define the functions common to the vehicle's tasks. This approach offers the promise of versatility and reconfigurability once the common set of abstract functions is developed.

2.1.2.1. The Naval Postgraduate School's Architecture

Researchers at the Naval Postgraduate School have developed the NPS AUV II [1]. It is a relatively small (380 lbs) vehicle which has been tested in a swimming pool environment, which eases some of the complications of sensor and navigational uncertainty, yet makes maneuverability and control issues more important than they might be in a less confined environment. The static, well-defined nature of the environment allows mission planning and modeling to be accomplished off-line and ahead-of-time. The off-line plan provides the real-time autonomous elements of the architecture with mission commands consisting of waypoints.

The onboard architecture is subdivided into vehicle functions such as guidance, navigation, and obstacle avoidance. The guidance system accepts geographic waypoints and task data as commands, and translates them into sequences of control commands to the autopilot systems. The autopilot systems command the vehicle actuators, using information fed back from the vehicle's sensing and navigation systems. The obstacle avoidance decision maker uses information from the pattern recognition system to command the guidance system to avoid collisions, and so forth.

The performance of the architecture can be evaluated directly through observing the vehicle's actions; if the vehicle runs into something, one need only examine the inputs and outputs of each of the subsystems to determine where the fault occurred. Similarly, the obvious connection between the structure of the architecture and the tasks which the vehicle performs simplifies the implementation of the architecture. The major drawback of this type of architecture is the difficulty involved in any modifications to the desired vehicle behavior. Should the vehicle be operated in a lake with a gently sloping bottom instead of in its swimming pool with well-defined sides, it would probably need to be reprogrammed to keep it from running aground.

2.1.2.2. The Linköping Architecture

The software designed by three graduate students at Linköping university consists of a three-level hierarchy for controlling autonomous systems [15]. The architecture is independent of the specific vehicle functions which are being performed; execution of any function involves all levels of the hierarchy. The hierarchy distributes activities with a short response time in its bottom level, and mission-planning and decision activities which have long response times reside in the top level. The middle level serves to interface the top and bottom levels.

The top level of the architecture operates in Executable Explicit Temporal Logic (EETL), a language based on the rules of reason and logic. Vehicle and environment models, goals, and actions are all described in logic statements, and the language searches

the logic statements for a series of actions to accomplish the goals. The series of actions is passed to the middle layer, where it is translated into sequences of lower-level activities for execution. Information about the state of the environment comes in to the top level from sensor monitors in the bottom level. This information is processed in the same manner as are the goals: the top level searches its knowledge base to infer abstract meaning from the sensor information.

The generality of this architecture facilitates modifications to the autonomous system software. The software itself is just as capable of reasoning about social etiquette as it is the world of the AUV; it applies the same rules of logic in both situations, but the rules operates on a different set of logic statements in each case. The drawback to this generality is that very specific concepts such as the idea that activities consume resources (fuel, time, and so forth) must be explicitly formulated in EETL, whereas a slightly less general architecture would probably make consideration of these concepts implicit and would be able to use efficient, special-purpose techniques for solving some of its problems.

2.1.3. Recognition-Based Modeling versus Simulation-Based Modeling

In recognizing and assessing the present situation, an architecture may use simple static models of its environment or it may use simulations of the environment, the vehicle, and even the software itself. With simulation, constituent laws of the simulated objects are used to extract more knowledge about the past, present, or future of the situation. In either case, models may be deterministic or may embody some measures of uncertainty such as those of classical probability.

Simple recognition models, such as those used in template matching, require less processing than do simulation models, easing timing requirements on the rest of the system. They can work well when used to recognize a limited number of situations in simple missions. Unfortunately, these simple models may not scale well to complicated

missions since their implementation requires an a priori determination of the situations they are required to recognize. The dimension of a recognition problem increases with the number of variables involved, rapidly becoming unmanageable.

Simulation models are computationally intensive, but they have the advantage of generality. Ideally, a simulation models the natural laws of its subject, and these natural laws apply in any situation. A simple recognition model can be applied to the end product of simulation, where it only need recognize a few situations. In effect, simulation may be used to extend the applicability of recognition models. If the recognition model were used alone, it would need to recognize not only those few situations which it recognized in combination with the simulation, but also all situations which lead to those few situations.

2.1.3.1. ISE's Simulation-Based Problem Modeling

The models used in the fault identification section of the architecture described by ISE in [13] are principally simulation-based; constraints such as feasible trajectories of the vehicle are developed from mathematical relations and past sensor data to generate likely future sensor data. A recognition-based model is used to decide whether discrepancies between the predicted observations and the actual ones are small enough to be acceptable or not. Acceptably small discrepancies are used to calibrate the model, and large discrepancies are used to signal the likelihood of either a failure or an unmodeled phenomenon. The results of the recognition model are passed to the decision and control structure for appropriate action.

The difference between the results of this approach and those of a purely recognition-based approach are subtle. The simulation approach determines whether or not the vehicle's models accurately represent the situation. The recognition approach to fault detection determines whether or not a problem modeled by the designers of the fault detection software has occurred.

2.1.3.2. Sea Squirt's Original Collision Modeling

The Draper-MIT Sea Squirt originally employed a simple recognition model to predict collisions. It simply assumed that if an object were directly ahead of it, a collision was imminent. While this worked well for the standard situation in which the vehicle was traveling forward, it was overly conservative and tended to prevent the vehicle from planned approaches to objects even at safe speeds. A natural improvement to the model was to predict a collision whenever the vehicle was moving toward nearby objects at some speed. That model is still unable to predict collisions in which the vehicle plans to turn toward the obstacle once it has come alongside it. This example illustrates the problem with extended recognition-based models; it is difficult for the designers of a model to predict every situation which could lead to the situation which needs to be recognized.

2.1.4. Knowledge Accessibility

In some architectures, an information system maintains and builds a knowledge base which is accessible to all of its elements. In others, each element or function of the architecture maintains its own knowledge base. A globally accessible knowledge base has the advantage of completeness and consistency; all knowledge gathered or created by the system can be incorporated into the knowledge base to make it as accurate and complete as possible. However, the global knowledge base also requires extra computational power to fuse together the various sources of information.

Systems which use multiple, local knowledge bases require less computation to deal locally with different types of information, but they may be less able to correlate the different types of information across local knowledge bases, possibly leaving different parts of the system with different and potentially conflicting views of the situation.

2.1.5. Strictly-Structured Storage versus Flexible Storage.

Some architectures make use of information which has a predetermined structure

and content, while others allow these to vary as the mission evolves and as plans and gathered information are updated. The fixed-structure approach is simple and efficient for routine tasks which rely on information of an inherently predictable nature. Representing information with variable structures is useful when the information may have varying content, as often occurs with highly abstracted information or information which cannot be efficiently fit to a fixed structure, such as map data which is sparse in some geographic locations and highly detailed in others.

Since interfaces and the information which passes through them largely define a system, a system's interfaces can prove useful in categorizing its architecture. The terminology employed in the definitions of the categories above differs slightly from that used in some of the papers surveyed. Therefore, it should be noted that in the preceding, the architectures are categorized according to their interfaces and to the preceding definitions rather than according to any classifications given by the papers' authors.

2.2. ANALYSIS METHODS

In order to determine the characteristics which contribute to the performance of an onboard vehicle software architecture, we need to understand what constitutes a good software architecture design. We define a good design to be one which is able to accomplish the vehicle's mission in the expected operational environment and which is also capable of salvaging the vehicle and/or elements of its mission under conditions which are abnormal or unexpected. Of course, this assumes that the capabilities of the vehicle's sensing and actuation subsystems are commensurate with the achievement of its mission goals.

The capabilities and features required of a good onboard vehicle software architecture are listed in Table 1 below. The vehicle must be able to reach the mission site, perform the mission, then return to a rendezvous point. It is also desirable for the

vehicle to be able to attempt multiple goals, to reason about the merits of each goal should they not all be attainable or not be known to be attainable, and to decide intelligently what should be done in the case of problems caused by a failure within the vehicle hardware, unexpected conditions in the environment, or other difficulty attempting to achieve a goal. Ease of communication with the operator improves the vehicle's cost effectiveness by reducing operator errors and programming time. Finally, real-time response must be adequate to effectively respond to unexpected events and to ensure vehicle safety.

Many design details must be resolved in order to satisfactorily achieve each of these capabilities or features in an autonomous vehicle: from the design of a control system to a clear, logical user interface. The nature of the basic abilities of the system, such as guidance, navigation, obstacle avoidance, and so forth, are determined by the particular algorithms employed to perform them. However, the overall software architecture constrains the type of algorithms that may be employed in these functions, in that the architecture determines the types of information they receive, how comprehensive that information is, and how the outputs of the algorithms are used. None of the desired AUV capabilities and features of Table 1 is achievable without a software architecture which supports and coordinates the vehicle functions effectively. The thrust of this paper is to identify those characteristics of a software architecture which allow the vehicle to exhibit these features.

TASK PERFORMANCE
- *ABILITY TO REACH TASK SITE*
  - *NAVIGATION, GUIDANCE, AND CONTROL*
  - *GEOGRAPHIC / OBSTACLE REASONING*
  - *CURRENT / ENVIRONMENT REASONING*
- ABILITY TO PERFORM TASK
  - *ADVANCED RECOGNITION*
  - *FINE LOCATING / MANIPULATING*
  - *REASONING FROM EXPERIENCE*
- ABILITY TO RETURN HOME

ACHIEVEMENT OF MULTIPLE GOALS
- INTELLIGENT SCHEDULING OF GOALS
- IDENTIFICATION AND REASONING ABOUT FAILURES:
  - *IN GOAL ACHIEVEMENT*
  - *IN VEHICLE HARDWARE*
  - *IN ENVIRONMENT MODELS*

REACTIVITY
- ADEQUATE REAL-TIME RESPONSE:
  - *TO NORMAL, EXPECTED EVENTS*
  - *TO EVENTS REQUIRING REPLANNING*

SAFETY
- PREDICTION OF FUTURE
- RECOGNITION OF BAD SITUATIONS:
  - *IN PREDICTIONS OF FUTURE*
  - *IN PRESENT*
- CONTINGENCY PLANNING

EFFICIENT COMMUNICATION:
- OF GOALS TO VEHICLE
- OF PROBLEMS TO OPERATOR

FLEXIBILITY OF COMMUNICATION: TIME REQUIRED TO DESCRIBE:
- A NORMAL MISSION
- A NOVEL MISSION WITH UNUSUAL REQUIREMENTS
  - *(e.g. don't stir up bottom sediments)*

COST EFFECTIVENESS
- EFFICIENT USE OF HARDWARE
- LOW PROGRAMMING COSTS PER MISSION
- LOW COST OF OBTAINING RESULTS AFTER MISSION

Table 1: Capabilities and Features of Onboard Software Architectures

3. DESIGN CONSIDERATIONS

Accomplishment of the objectives of AUV's, as with the accomplishment of the objectives of any autonomous system, may be considered as two separate problems. The first is that the autonomous system must create a plan to achieve its objectives, and the second is that it must execute its plan. Replanning may be necessary should conditions in the system's operational environment at the time of execution not match the conditions expected during planning. The replanning is performed based on the state of the vehicle and environment known at the time of replanning. This approach of beginning with a plan, or open-loop sequence of commands to achieve system objectives and following that sequence of commands until conditions in the environment of the system change, is similar to the control methodology of *open-loop feedback control* [20]. In some form, all of the AUV approaches surveyed for the summary in Chapter 2 above follow this control methodology, with the possible exception of the pure form of Rodney Brooks' layered control.

The two phases of autonomous operation, planning and plan execution, can be posed as optimization problems. The objective of the plan optimization problem is to create plans that are feasible and that achieve the system's objectives in the best possible way, maximizing the compliance of the plan with the system's objectives while minimizing resource use (money, fuel, time, etc.). Plan optimization comprises the open-loop part of the open-loop feedback control. The objective of the plan execution optimization problem is to follow the plan as closely as possible (minimize deviation from the plan), and to coordinate the process of plan optimization that is required to develop a new plan when it is no longer feasible to follow the current plan. Plan execution is therefore the feedback part of the control.

Classical guidance and control systems are an example of open-loop feedback

control; the guidance system creates open-loop trajectories for the control system to follow. The trajectories created by the guidance system begin from the current state of the controlled system. The guidance system may be considered a specialized type of plan optimization system in which the plans are represented as trajectories. The combination of the control system and the controlled system's sensors correspond to a simplified plan execution system. The control system typically minimizes deviations from the trajectory created by the guidance system, where the cost of deviations is modeled in the control system's objective function. Since the guidance and control system is an autonomous system, it is not surprising that it matches this plan-based decomposition of autonomous systems.

3.1. OPTIMIZATION BACKGROUND

A significant theoretical background in problem-solving techniques has been developed for mathematical optimization theory [19],[21],[22]. A typical approach to solving a large optimization problem is to decompose the problem hierarchically, such that the top level (master level) of the hierarchy coordinates a number of inferior optimization problems. The inferiors solve less complex problems independently and return the results to the top level, where the solutions from the inferior levels are used to coordinate the next iteration. The particular method of coordination employed affects the nature of the sub-problems and determines the class of optimization problems which are solvable.

3.1.1. Feasible approach

The feasible decomposition for hierarchical optimization methods insures that the overall problem solution meets its constraints at every iteration [19]. It does so by having the superior level of the hierarchy constrain the outputs of the inferior levels to the class

of feasible solutions. The inferior levels must then produce a solution within this class of outputs. For problems in which the nature of the subproblems is such that there exists a solution to the subproblem which will create any desired output of the subproblem, (i.e., the subproblem is controllable) this method works very well. However, there exist a great number of optimization problems for which the subproblems are not controllable, in which case one of the other coordination methods is preferred.

3.1.2 Infeasible approach

In the infeasible approach, the master level coordinates the lower levels by setting "prices" on the problem's constraints. The prices are included in the objective functions of the lower level problems and indicate to the lower levels the cost of violating each constraint. The upper level sets these prices in an attempt to guide the solutions of the lower level problems to meet the overall constraints.

Each constraint in an optimization may be either an equality constraint or an inequality constraint. An equality constraint is a requirement that a function of the solution variables must be exactly equal to a specified value for the solution of the optimization problem to be feasible. An inequality constraint requires a function of the decision variables to be either less than, or less than or equal to, a specified value. Any constraint can be posed such that only one side of the inequality or equality is not constant; the quantity on this side of the constraint expression is the *constrained quantity.*

If an inequality constraint is violated, the upper level raises its price so that the lower levels conserve the constrained quantity. Conversely, if an inequality constraint is not violated, its price is lowered so that the lower levels may use more of the constrained quantity if it is beneficial to do so. The lower levels take these prices into account by subtracting from their objective functions the price of each constraint times the amount of the constrained quantity they use.

Similarly, if an equality constraint is violated, the upper level raises its price to

encourage the lower levels to meet the constraint. The lower levels take each inequality constraint into account by subtracting from their objective functions the price on the constraint times the difference between the constrained quantity and the value it is constrained to equal. At the solution to the optimization, the equality constraints' prices should be equal to the cost associated with changing the constraint, so that the constraints are met.

3.1.3 Mixed approach

The mixed approach takes advantage of features of both of these approaches. It employs the feasible approach with some of its subproblems and the infeasible approach with others. In this way, the controllability of some of the subproblems may be exploited, without sacrificing the solution of the uncontrollable subproblems.

3.2. OPTIMIZATION-BASED EXAMINATION OF ARCHITECTURES

The architecture characteristics described in Chapter 2 each contribute to or detract from the potential performance of the planning software as an optimization-based problem solver. Decomposition of the planning problem into a hierarchy of subproblems has the potential to reduce the effort required to solve the optimization problem considerably. Most practical algorithmic approaches to problem solving are polynomial. That is, the effort required to solve the problem is on the order of n^k, where n is a measure of the size of the problem being considered and k is a constant related to the type of algorithm used, typically between 2 and 4. Decomposition of a problem of size n into m problems of size n/m has the effect of reducing the computational requirement to $m*(n/m)^k$, a factor of $(1/m)^{k-1}$ of the requirement of the original problem.

Of course, decomposition of the problem into too many subproblems may cause the subproblems to produce solutions which are not a good solution to the overall

problem when taken together. Alternatively, if the subproblems are properly coordinated so that they do produce a good solution to the overall problem, the computation required for the coordination may exceed the computation required for a decomposition which uses fewer subproblems. One cannot simply divide a problem of size n into n pieces and expect the solution to the problem to require computational effort on the order of n. The problem may only be decomposed into subproblems if the solutions to those subproblems are largely independent of each other.

The heterarchical approach to an architecture could theoretically have the same advantage in required computation as the hierarchical approach, if the overall problem were such that it could be divided into *completely* independent parts. The heterarchical approach is a decentralized approach to problem decomposition; there is no coordination among the subproblems. In other cases, the heterarchical approach provides sub optimal solutions, for the heterarchical approach has no formal provision for coordination between the elements of the heterarchy. Although there are few situations in which a problem may be divided into completely independent subproblems, specific cases do exist for which heterarchical control is very attractive. One of the most successful uses of heterarchical control so far has been as a "safety" layer for autonomous vehicle architectures. In this capacity, the safety layer's purpose is to ensure that the immediate risk to the vehicle remains below a pre-specified level. When required, a course of action which poses the least immediate threat to the vehicle is selected. Different courses of action are postulated by the heterarchy's elements, each of which considers the risk to the vehicle due to one particular type of threat such as the threat of collision or of exceeding the vehicle's depth rating. The threats examined are modeled to be completely independent of each other, although in reality they may not be. The arbitrator selects the course of action which minimizes those threats, which in most cases is the desired mission command. This approach to vehicle safety may fail when the threats to the vehicle in a given situation are not independent; for example, avoiding an obstacle may

cause the vehicle to run aground or to exceed its depth rating.

To develop an architecture which is generally applicable, one must allow some interdependence between the subproblems, a consideration which suggests the use of a hierarchy. In addition, to make the hierarchy applicable to a wide variety of autonomous system problems, its structure must not be rooted in a decomposition of any particular system, but should be based on a decomposition which may be applied to any of the systems for which it may be applied. The tasks that are specific to a particular application must be decomposed according to the architecture's structure when the architecture is configured for that application. The objective function of the optimization problem must be separable for the theoretical hierarchical improvements in computational effort to be realized.

3.3. DIMENSIONS OF REAL-TIME PLANNING PROBLEM

A number of aspects of the real-time planning problem can be analyzed in the context of optimization theory. The traditional static planning problem, which is one dimension of the dynamic, real-time problem, may be decomposed into a hierarchy which has abstract, long-range plans at the top levels and detailed, short-range plans at the lower levels [18]. The top level calls upon its inferior levels to solve the specifics of the planning problem. Feasibility of the plans is achieved when the autonomous system's resource budgets are not violated by the plans that have been created.

The uncertainty in the future states of the vehicle and mission environment for any planning problem complicates the application of the static planning hierarchy as described above. In the presence of uncertainty, all of the subproblems of the global problem may not be solvable at the same time, since some require information which may not be available until others have been executed. Some of the subproblems may be solved using predictions or estimates of information which can't be verified until after the

autonomous system actually begins execution of the plan. In these ways, the execution and the planning processes of the autonomous system are tightly coupled, leading to the second dimension of the planning problem as described below.

The second dimension of the real-time planning problem is the regulation of the planning process when viewed as an operation that occurs in real time, that consumes resources (computational operations, computer memory, power in some applications, etc.) and that is capable of producing value [21]. The coordination and allocation of the computational resources to be used at each level of the planning system hierarchy is a problem which must be solved outside the time-horizon based hierarchical decomposition. In other words, the planning system may be viewed as an optimization problem of two levels: the bottom level which creates the plans and the top level which contains the planning management algorithms that control and coordinate the planning operations of the bottom level. The bottom level is itself a hierarchy, each level of which is characterized by the spatio-temporal extent and level of detail of the plans and information created and used at that level. Thus, the top level of the two-level *management hierarchy* allocates computational resources to the subproblems of the bottom level. The subproblems of the bottom level each represent one level of the temporally decomposed *temporal hierarchy* of planners as depicted in Figure 1.

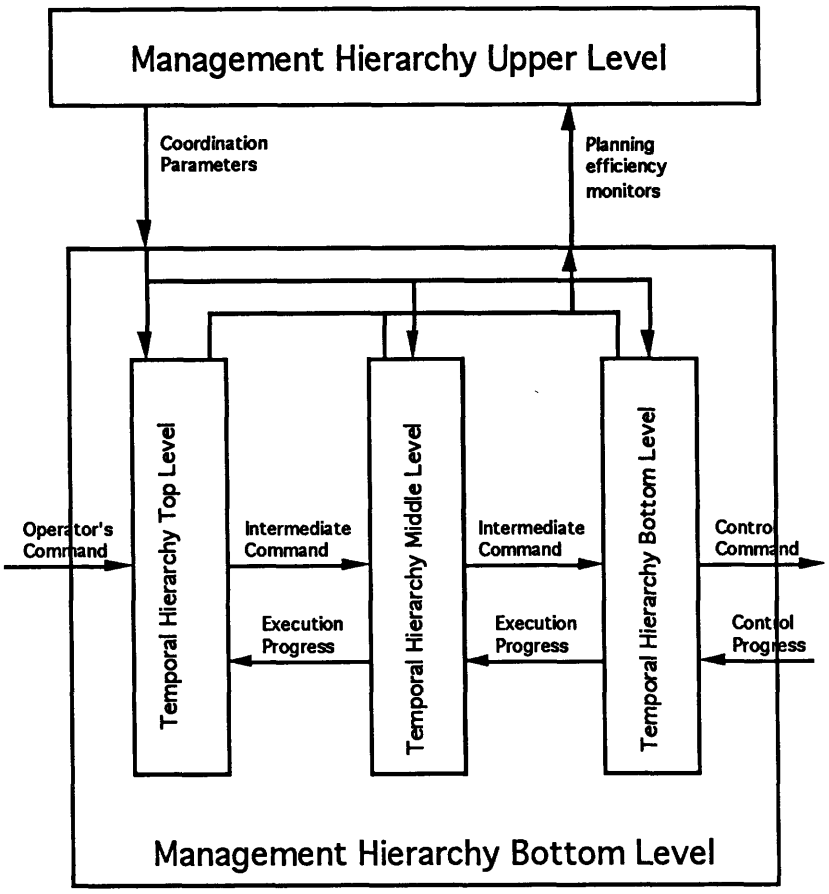

Figure 1: Dual Hierarchy

3.4. OBJECTIVE FUNCTION DECOMPOSITION

In order for the results of the analysis that is applied to traditional hierarchical systems to be applicable to the planning system, the planning system's objectives must be quantified in a manner that is separable, i.e., the global objective for the system, F, must be monotonically dependent on the objectives of the subsystems:

$$f \leq f' \Rightarrow F(f) \leq F(f')$$

where f is any lower-level objective function. That is, a change in the solution to a subproblem which improves the lower level objective function must also improve the

global objective function. This relation holds in particular for the case where the overall objective function is the sum of the lower-level objectives:

$$F = \sum_{i=1}^{N} f_i$$

It also holds for the case in which the overall objective function is the product of the lower-level objectives,

$$F = \prod_{i=1}^{N} f_i$$

provided that the f_i are all positive. [22]

3.4.1 Temporal Decomposition

For an autonomous system, the overall objective might be stated in terms of monetary profit, scientific value of the data collected, or some other abstract quantity. This quantity, or *value* of a mission is generally divisible among the different *goals* of the mission in an additive manner. In this way, the objective of the mission may be stated as an optimization problem:

$$\max \sum_{i=1}^{N} V_i$$

$$\text{s.t.} \sum_{j=1}^{N} res_j^k \leq RES^k, \quad \text{for all } k$$

where V_i is the expected value of goal i (the value of goal i times the probability that goal i is successfully completed) and res_j^k is the expected use of resource k by goal j. RES^k represents the total amount of resource k which is available for use during the mission. Thus, the mission's objective function is separable into parts associated with each mission goal, and the coordinating constraints are established through the system's resources. Note that there may also exist coupling constraints between the goals, but that these are evaluated in the value of the goal: a goal may be considered to have no value if any of

the requirements placed on the autonomous system's state at the beginning of its execution are not met, in which case the probability of success of the goal is zero, causing its value to be zero (see Figure 2).

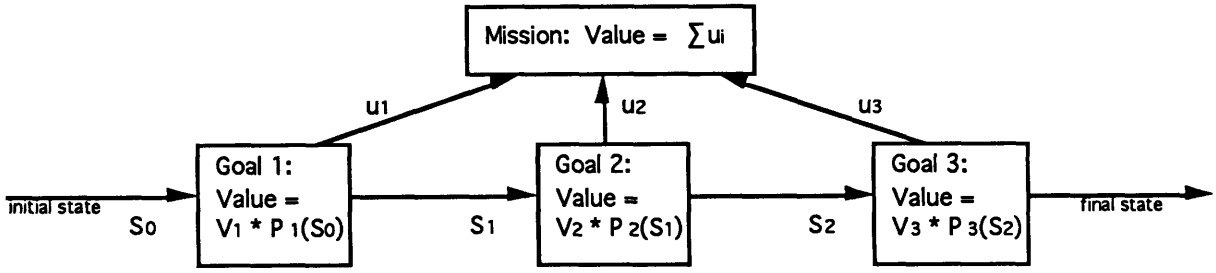

Figure 2: Coupling Constraints and Decomposition of Value

According to the decomposition of the planning problem described above, each of these mission goals must be partitioned into smaller problems of higher detail, or subgoals. As with the division of the mission into goals, the division of a goal into subgoals requires that the goal's objective function be decomposed into objective functions for the subgoals. Additionally, the coupling constraints of the goal must also exist at the subgoal level, since the subgoals represent the goal but at a finer level of detail.

In a hierarchical solution to an optimization problem, the objective function of each subproblem is equal to the top level's objective function modified by a coordination parameter that is determined at the top level. For example, in the infeasible method, the lower-level objective functions are modified by the prices on the constraints. In the application of hierarchical optimization theory to the planning problem, the value of the objective function of a given level is referred to as the *utility* of the level, which differs from the expected value of the plan at the level.

3.4.2 Management Hierarchy Decomposition

The objective of the top level of the management hierarchy, or *manager*, is to achieve the full potential of the temporal planning hierarchy. In other words, the

manager's objective function must be some combination of the objective functions of the levels of the temporal hierarchy. A simple objective function might be

$$F = \text{argmin}(f_i / n_i)$$

where F is the manager's objective function, the f_i are the objective functions of the levels of the temporal hierarchy, and the n_i are values against which the f_i are normalized.

The constraints on the manager are that the total computational resources expended in each interval of time must be less than or equal to the computational resources available for that interval. Because the manager determines when to allocate computational resources to each level of the temporal hierarchy, it must evaluate the objective functions of the levels of the temporal hierarchy. As the mission unfolds and unexpected events occur, the solutions created by the levels of the temporal hierarchy will degrade. The manager must compare the solutions created by the temporal hierarchy to the observed results of executing those solutions to determine whether or not improvement of the solutions is possible with more computational effort. If improvement is possible, the manager must determine how much computational resources are justified by the potential improvement.

3.5. GENERALITY-SPECIFICITY OF DESIGN

Based on the examples in Chapter 2 above, architectures that are general in nature, such as the Linköping architecture, tend to be inefficient in that many facts about the AUV problem which can be taken for granted in other architectures need to be represented explicitly in the generalized architecture. On the other hand, architectures that are arranged rigidly in terms of the vehicle's functions are not likely to be useful for applications other than for the exact vehicle and mission for which they were designed. What is needed is a simple way to implicitly incorporate the assumptions common to all autonomous planning problems into an architecture which is general and flexible enough

to be useful for different types of missions and vehicles.

Presented below in Chapter 4 is one such architecture. It is assumed that any activity may deplete the system's resources, contribute to the value of the mission, and change the system's state. Each level of the hierarchy breaks down the planning problem given it by its superior in order to best meet the superior's expectation that certain goals will be accomplished within the allocated resources.

The architecture is general in the sense that its optimization, plan-generation, planning management, and resource use estimation functions can be applied to many planning problems. The planning problem is defined for these general algorithms through models of the resource use of the activities associated with any solution to the problem and through the hierarchical decomposition of the problem. The architecture uses these models to evaluate the plans it creates, simulating the plans based on the provided models to evaluate the utility of each plan and possible improvements to it.

It is assumed that the architecture will be employed in a framework which includes a fault-detection and performance-modeling system and a central information-processing system, as well as a control system capable of controlling the autonomous system. The plans from the planning system will be used to generate controller commands directly, and the results of the fault detection and performance-modeling system will be used to update the vehicle models, as shown in Figure 3.

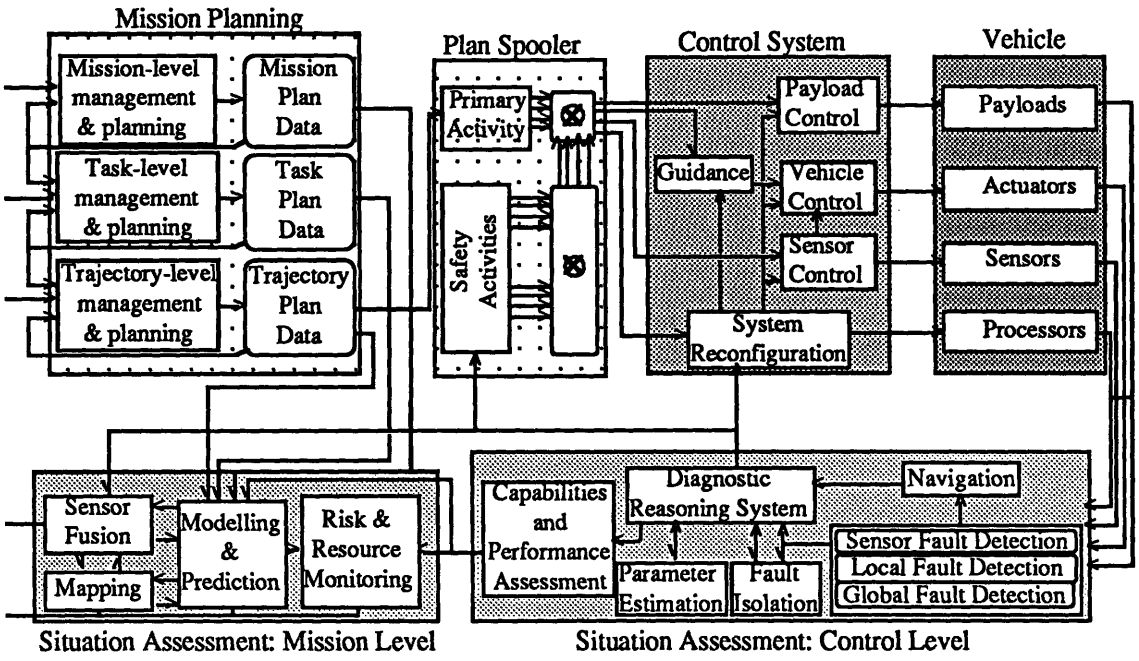

Figure 3: Interactions Between Mission Planning System and Autonomous System Software.

4. OVERVIEW OF PLANNING SYSTEM ARCHITECTURE

The proposed planning system architecture is hierarchical and consists of three levels of planning: the top level creates plans spanning the entire mission at a low level of plan detail, the second level creates plans over a shorter time horizon but at a higher level of plan detail, and the third level plans only over the short-term future but at a level of plan detail sufficient to completely define the vehicle activities over that time span. This architecture is typical of the classical hierarchical system decomposition described in [17]. The complete set of functions required for the *real-time* implementation of this hierarchical solution in the context of an autonomous system is described below. The implementation of each of those functions is described in the remaining chapters.

The proposed hierarchical planning architecture is applicable to a wide variety of systems. At the heart of each level of the planner is a set of functions which optimize plans, monitor plan execution, replan when problems occur, and manage uncertainty. The optimization of plans considers both the use of resources and the completion of goals. A significant part of the effort required to customize the proposed hierarchical planning system architecture to a particular application is in developing models which describe the system's functions, their resource uses, and their effects on the system's state. For the autonomous underwater vehicle application considered here, these functions include activities that range from maneuvering and vehicle control (at the bottom level of the hierarchy) to surveying regions of the sea floor and locating and identifying objects of interest (at the top level). The effects of these activities on the vehicle's resources (e.g., fuel, time, risk, reliability) must be included in those models. Finally, for any real-time system, the use of finite computational resources is an important auxiliary planning problem, since vehicle activity planning depletes computational resources which must be shared with sensor processing, control, and other subsystems.

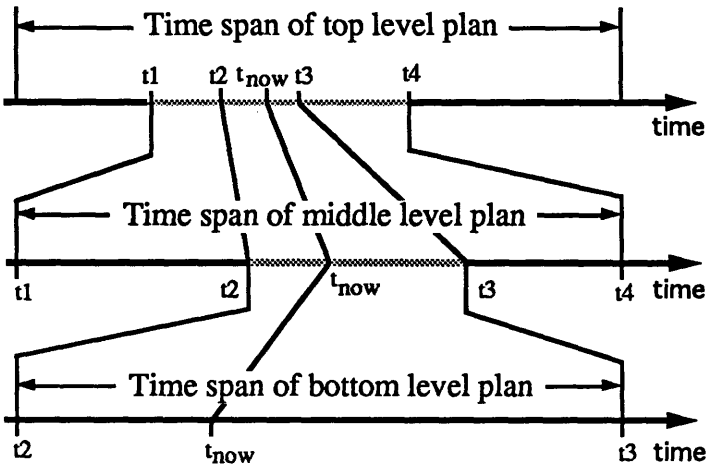

Figure 4: Hierarchical Temporal Decomposition

Figure 4 shows the temporal decomposition of the activities associated with each level of a three-level hierarchy. In that figure, the grey portion of each time axis is expanded on the next axis down. The top level maintains a plan of the entire mission and enables the system to make decisions which are consistent with long-term goals but which may not be attractive from a purely short-term perspective. Its plans are defined in terms of abstract goals provided by the user which contain little detail. Since detailed planning of the mission far into the future is likely to be futile due to uncertainties in vehicle and environment models, planning in terms of probabilistic models and abstract goals is used to increase the chance that the top-level plan remains viable over a long span of time as the mission is executed and unpredictable events unfold. It can be argued that in this way the system makes decisions in a manner similar to that of a human operator.

The middle level takes commands from the top level, treating them as goals to be accomplished. The middle level devises a plan to accomplish those goals within resource constraints that are passed down from the top level, and then it passes that plan down to the bottom level as a command. In reality, the middle level is no different from the other levels; the algorithms which accomplish the functions within the hierarchy are identical at each planning level, so the functional distinctions between the levels of the hierarchy come about through their relative positions above or below each other. The planner level

defined here would work equally well in a hierarchy composed of two or ten levels.

The primary purpose of the bottom level is to flesh out the details necessary for generating sensor and actuator commands that ultimately result in executing the plan. Thus, the results of planning at this level drive the autonomous system's controllers directly. Since information at this level regarding the vehicle and its environment should be detailed and accurate over the horizon of the plan, the level doesn't plan very far ahead. In order for its plans to stay ahead of the controllers, planning must be performed quickly, efficiently, and frequently.

The number of levels appropriate for a given system depends on the complexity of the missions to be performed and the speed at which the onboard system must be able to replan. A system which has many levels will be able to plan complicated missions, but it may also require that a large amount of computational resources be devoted to coordinating between the levels. Conversely, a system with fewer levels will be able to react more quickly, but it will be less capable of planning a complex mission. The requirements of any potential implementation of this planning architecture must be analyzed to decide what size hierarchy is needed.

Each planning level uses information at a different level of abstraction in making its decisions. Two basic requirements determine the information needs of a level. First (1), each level must have information which spans the spatial and temporal horizons of the plans that it creates. Second (2), the information about the mission environment at the far reaches of the spatio-temporal scope of the plans may change as onboard sensor information is gathered, but updates to the information within this scope should ideally be accomplished far enough ahead in time to allow the system a chance to successfully replan, if deemed necessary, in order to accommodate the updated changes in state. This second requirement imposes constraints on the design of the platform and its sensing systems, since the time it takes for the updated information to reach the planning level effectively reduces the time a given level has available to replan in response to those

updates.

At each level, models of the vehicle and environment that are used in planning should be commensurate with the level of detail of the plans generated. For example, large-scale maps are used at the top level; these contain information over the entire mission area, allowing the entire mission to be planned consistent with the first requirement above, and are updated quickly relative to the fairly long replanning time of the level, helping to meet the second requirement. The middle level uses local representations which are more detailed but require less sensor data and information fusion to create than do the highest level's maps, providing the information required more quickly since there is less time to replan. At the lowest level, real-time sensor data is used to augment a priori maps and models to provide detailed, accurate information about the vehicle and its immediate environment.

4.1. THE PLANNING LEVEL

Each level of the hierarchy is subdivided into the functions shown in Figure 5. Commands which enter a level are processed by the ***Level Manager***, which stores them in its goals buffer. When the global manager (the top level of the management hierarchy) decides to replan at a level, the level manager at that level passes some of the goals to the ***Goal Planner*** as described below in the section on the manager. These goals are represented within the goal structure as described in the following. The goal planner creates a plan from the goals and optimizes it according to the goals' values that have been sent down in a command to that level from the planning level above. The goal planner then passes the results back to the level manager. In order to optimize the plan, the goal planner must be able to evaluate the plan in terms of both its utility and its resource requirements. The definitions of a plan's utility and of its resource requirements are stated below. ***Resource Estimators*** determine the resources used by each type of goal

in a plan.

The level manager updates the current plan for the given level with the new plan created by the goal planner and instructs the ***Activity Planners*** to create subgoals for each of the plan goals. The subgoals serve as the commanded goals for the next level down in the hierarchy. The level manager then passes the subgoals down to the next planning level (its inferior) and reports its new plan up to its superior planning level. If the planning level were the bottom level, its subgoals would be passed to the control system. If it were the top level, the new plan report feedback would go either to an operator (if one is on-line or available via communication) or into storage, depending on the degree of autonomy of the system.

When an inferior creates a new plan, the level manager at its superior level uses feedback of the reported new plan to update its estimates of the value of the plan and resource requirements for the plan. Those updates reflect both the feasibility of the plan over its complete temporal horizon and the current progress of the plan's execution relative to expected progress. Should the global manager determine that the superior level's plan is not being followed properly due either to vehicle failures, to unexpected problems with its execution, or to modeling errors in the resource estimators, it will command the level manager to initiate replanning at its level. The details of the mechanisms which cause replanning to occur are discussed in detail below.

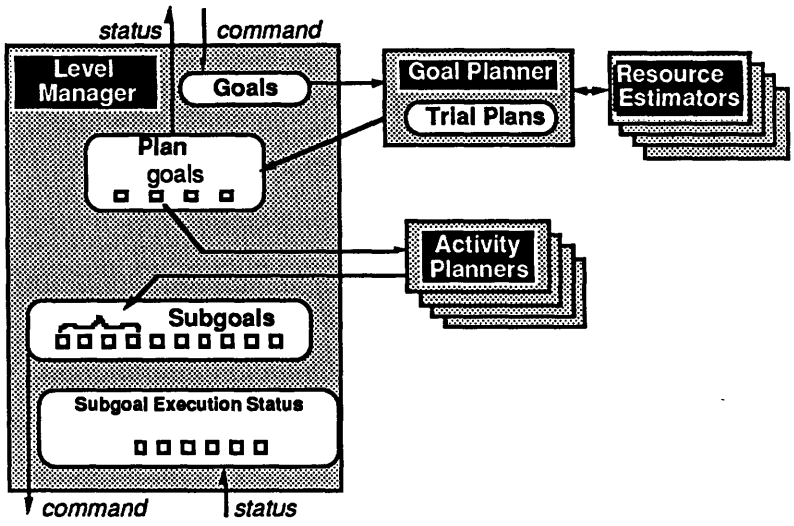

Figure 5: Elements of a Generic Level of the Planning Hierarchy

4.2. PLAN STRUCTURES AND GOAL STRUCTURES

Plan and goal structures are ordered lists of goals that can be decomposed into subgoals by the activity planners of a single planning level. The principal form of communication between planning levels occurs through plan structures and goal structures. A plan structure contains a plan, whereas a goal structure contains a goal list. The only difference between a plan and a goal list is that the goals which make up a plan have been ordered and optimized by the goal planner, and the utility and resource use values associated with the plan's goals are those *expected* to be achieved. In contrast, the goals making up a goal list were created and passed down by the previous level, and the associated resource use and utility values represent the resource allocation constraints to be satisfied and the desired utility to be achieved by plans produced at the lower level.

A higher level commands a lower level by giving it a goal structure that has been derived from its subgoals (see Figure 5). The lower level translates the subgoals in the goal structure into a plan, which it stores in a plan structure. The lower level then returns its plan to the higher level (which stores the plan in Subgoal Execution Status as shown in

Figure 5) so that the higher level and the global manager may gauge the feasibility and effectiveness of the lower level's plan.

Since plans and goal lists are the principal means of communication between planner levels, their representation has a significant impact on the performance of the planning system. By allowing conditional branches in the plans and goal lists, high-level planners can plan for any of several situations likely to occur in the future. As those situations occur, execution follows the appropriate branch, obviating the need for replanning at the high level. This allows the planning system to function well in an uncertain environment.

The plan structure (the analogous representation for a goal structure is identical) is shown in Figure 6. Each oval represents a *node* of the plan which is a structure representing a vehicle goal. The nodes are connected sequentially, in order of execution, to form a plan. A *branch node* (see Figure 6) is a node from which more than one alternate plan segment or branch follows; the choice of the specific branch that is executed following a branch node will depend on the conditions present at execution time. Thus, this structure allows conditional branches and loops to be represented.

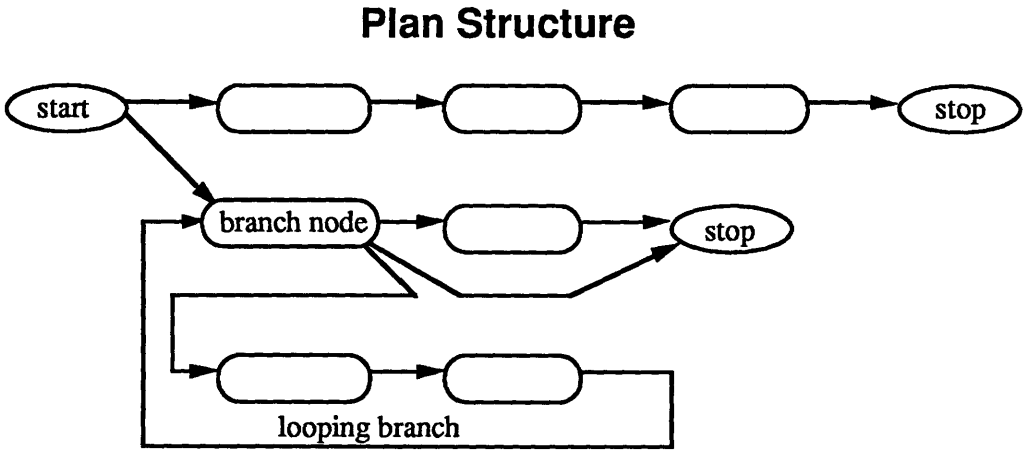

Figure 5: Plan Structure

The goal associated with each node in a plan or goal list has a goal ID, specifying which type of goal is associated with the node. The node also contains a parameter list

which contains additional information specific to the goal. For example, in the case of a transit goal type, the goal parameters would represent the location of the destination for the vehicle, whereas for a survey goal, the parameters would represent the area to be surveyed and some information about the accuracy required of the survey data.

Each node also contains a resource vector and a state vector associated with the node's goal. The resource vector is interpreted as described above. The state represents the expected state of the vehicle at the beginning of the node's execution. The state of the vehicle expected upon completing the execution of a node is stored in the expected (beginning) state of the following node or nodes. A plan's state and resource requirements are used to help determine its feasibility.

4.3. RESOURCE ESTIMATION

The user assigns a value to each goal which is input to the top level of the planning system. At lower levels, it is the job of the activity planners to distribute the value of their goals among the subgoals. At each level, the value is attached to the node containing the goal. The overall expected utility of a plan, then, is the sum of the goal values multiplied by the probability of success for each respective goal (i.e., the expected goal values) weighted by a penalty function:

$$\text{Utility} = \left[\sum_{i=1}^{\#\ \text{of goals}} \text{Value}_i \bullet P_i \right] - \text{Penalty(Res)}$$

Where Value_i is the value assigned to goal i, P_i is the probability that goal i will be accomplished successfully, and Penalty(Res), the penalty function, is a function of the normalized resource vector Res, the resource use divided by the allocated resource use. The penalty function serves to enforce a higher level's constraints associated with the resource allocations passed down as part of the higher level's command.

Because of the unavoidable uncertainties in both modeling and a priori information as well as the uncertainties in future vehicle activities associated with conditional branching in the plan structure as defined above, the utility is not deterministic. In order to calculate the utility of a plan, its resource use and the probabilities of goal accomplishment must be estimated. Thus, resources and utility are represented as expected values with an associated expected deviation, and plan evaluations made during the course of the search for the best plan are made on the basis of expected value and uncertainty. For a branch node, the activity planner computes a probability of executing each branch. Those probabilities are computed on the basis of the activity planner's probabilistic models of the branch event and the environment. Therefore, the resources and utility of a plan are determined as the probability of the execution of each branch times the appropriate expected value of the branch:

$$V_t = \sum_{i=1}^{\#\text{ of branches}} P_i \bullet V_i$$

where V_t is the total value of some resource or utility of the plan, and the P_i and V_i are the probabilities of execution and the values of each branch of the plan, respectively.

The deviations of these values are calculated as if the choice of which branch to follow were a random process independent of the branches taken before or after the branch; the variances of the resource and utility values of the plan following a branch node are taken to be the variance of that value of each of the branches times the probability that that branch is taken:

$$Var_t = \sum_{i=1}^{\#\text{ of branches}} P_i \bullet Var_i$$

The activity planners and the goals must be carefully defined to represent processes for which this is a reasonable approximation to the variance.

The resource estimators perform two functions. They propagate the plant's state forward through each plan node and they compute estimates of the resources required for

each node of the plan. The elements of the state are represented probabilistically in terms of their expected values and their variances, just as the resources are. When the plan reaches a state from which, according to its expected value and variance, it is unlikely that further progress can be made, the rest of the current branch of the plan is ignored and considered to have no value. In our implementation, it is considered unlikely that further progress can be made when there is a 98% chance that fuel or time constraints will be violated. In this way, the resource estimation doesn't waste computation time on plans which are unlikely to succeed. This mechanism also allows the resource estimation routine to truncate the evaluation of infinite loops without actually looping forever.

A loop occurs when a branch of the plan leads back to an earlier part of the plan (see Figure 6). Loops are created when a segment of the plan needs to be repeated an indeterminate number of times. The loop's exit condition is stored in a branch node which has at least one of its branches leading forward in the plan; when that forward-leading branch is executed, the loop has finished.

4.4. PLANNING MANAGER

The key to successful hierarchical planning lies in the approach taken in the management and coordination of the activities between levels. During nominal planner operation, new plans are created frequently at the low levels to replace the currently executing plans at those levels before the completion of their execution. Figure 7 shows a representation of this on a single planning level.

When a significant portion of the current plan has been executed, a new plan must be created or the system will run out of plan to execute. The manager has to predict (or specify) how long it will take to create the new plan (in Figure 7: *time required to replan*). Clearly, the point in time when the new plan will start cannot occur before the time when the process of replanning is completed. On this basis, the manager picks a

feasible start time for the new plan, and directs the goal planner to replan from the state expected at that start time. That expectation is based on the evolution of the state as governed by the execution of the current plan. At the selected start time for the new plan, the new plan will be spliced to the current plan, truncating the current plan, and execution continues henceforward on the basis of the new plan.

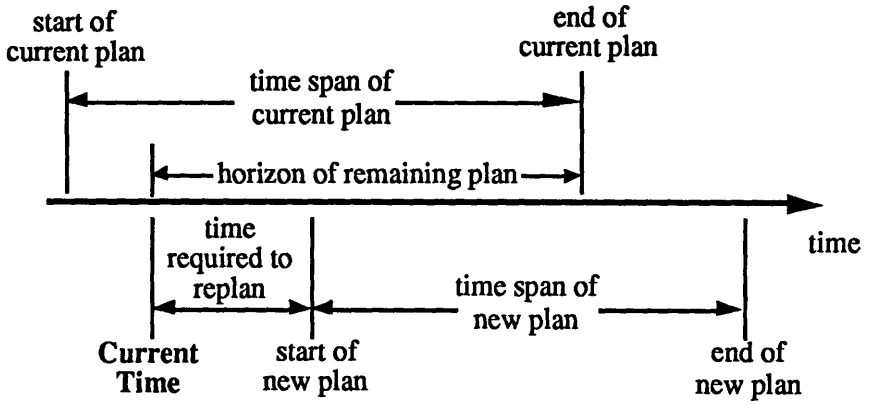

Figure 7: Splicing in a New Plan

Each level of the hierarchy contains a goal buffer, a plan buffer, and a goal planner. In order to initiate replanning, the level manager must supply its goal planner with the appropriate initial state and a set of goals to be optimized. The initial state is merely the state predicted at the start point (in time) for the new plan. The set of goals is taken from the goal buffer for that level; the number of goals to be included in the set is determined by the global manager on the basis of the desired plan horizon of the level and the temporal extent of each of the goals. The resource allocations for the goals are taken from the resource expectations in the goal buffer. The resource allocations are discounted by a factor which the global manager adjusts to keep the total resource use at that level close to the expected total resource use, while maintaining a reserve of resources to accommodate potential future shortfalls. The level manager uses the plan created by its level to generate goals for its inferior.

The global manager coordinates the activities at the various planning levels to

keep the entire planning system in balance and stable; to be in balance and stable, the planning system must have at each planning level a feasible plan which is of a temporal scope appropriate for the level, or it must be replanning to correct any detected problem. If the environment were completely predictable, coordination of the planning process would pose little problem, and the manager would only need to update each planning level as its inferiors began running out of goals to execute. However, it is inevitable that unexpected events will occur no matter how accurate the onboard models are. Thus, the planning system must be capable of accommodating problems that unexpectedly crop up during the plan's execution.

In monitoring the execution of the plan, the level managers at the various levels update the resource estimates and expected utility of their current plan at regular intervals. As the plan is executed, the updated resource estimates are likely to vary from the amounts expected at the time that the plan was created. Problems that arise which prevent a task from being accomplished will be detected because the resource use estimates will substantially increase or the expected utility will plunge, or both. The global manager uses these indicators to decide when a level should replan to accommodate an unpredicted problem.

In accommodating an unpredicted problem, planning typically propagates down from the level at which the problem in the plan was first detected; if the manager decides to replan at the top level, the revised plan is passed down to the second level which then replans using the new goals, and so forth. This sequence may fail at times, however. Should the bottom level run out of plan to execute before the replanning process above it is completed, it must replan using the (problematic) current plan's goals. Alternatively, should a problem occur when replanning is already in progress and the problem is such that it makes the new plan being created worthless, a mechanism must be in place to detect that problem, abort the current replanning activity and initiate a new replanning process. Thus, there are three classes of events which may cause the planning manager to

direct any given level to replan: (1) the level runs out of plan to execute, (2) the level's plan is no longer viable because of a new or unmodeled situation, or (3) the level has been handed new goals by it superior.

When problems from more than one class of events occur simultaneously, the manager evaluates measures of the relative importance of the three classes in order to decide how to respond. To this end, the manager quantifies the desire to replan at each level of the hierarchy due to each of the three classes, and combines these three quantities at each level into a single measure referred to here as 'replanning urgency'. The level with the greatest replanning urgency is chosen to initiate replanning, and the replanning urgencies on all levels are recalculated and the process repeats.

The importance of replanning due to running out of plan is defined as a function of the fraction of the desired planning horizon represented by the unexecuted time span of the current plan. The importance of replanning due to an unexpected problem with the current plan is related to the deviation of the resource use and utility from that expected of the current plan. The importance of replanning the next level down after replanning at a given level is dependent on the fraction of the goals of the old plan which are modified in the new plan. The precise calculation that is used to quantify each importance and to combine them into a single replanning urgency has a significant impact on the throughput of the system, and is under investigation. The formulation of the replanning urgency must be designed to ensure that the planning system is reactive enough to respond quickly to replanning needs while at the same time is stable enough to ensure stability of the planning process.

The sequence of execution of the managers follows the steps outlined below:

1) the level managers calculate the replanning urgency for each plan.
2) the global manager selects the plan with the highest replanning urgency.
3) the global manager selects the resource allocations and goals with which to replan, and estimate the beginning state (the state at the time the new plan will be put into effect).

4) the level manager directs the goal planner to optimize the selected plan.

5) the level manager creates the subgoals described in the optimized plan

6) the global manager updates the resource and utility estimates for each plan, starting with the lowest level plan and working up to the mission plan. At this step, new information about the present state is propagated forward through the plan according to the resource estimators' state propagations.

7) go to 1).

4.5. GOAL PLANNER

The purpose of the goal planner is to develop a near-optimal plan from a set of desired goals, ensuring that specified resource constraints are met. One of the resource constraints passed to the goal planner is the allocation of computational resources to be applied to the process of generating a plan. In order to ensure that the planning system does not exhaust its computational resource budget before a new plan is found, the goal planner employs an iterative heuristic optimization technique which produces a plan within the budgeted computer resources in such a way that the more resources are allocated the better the answer is likely to be [18]. The goal planner may be given an initial plan as a starting point (for instance the current plan when a plan update is requested), or it may generate the entire plan on its own.

The goal planner's optimization algorithm iteratively improves plans. During each iteration, the goal planner creates a trial set of candidate variations of a working plan and evaluates each candidate with a set of scoring functions. The scoring functions are used to probabilistically select the next working plan, with each candidate plan's chance of being selected as the next working plan being proportional to its score. After probabilistically searching the plan space for the allocated amount of planning time, the goal planner returns the best plan it has found.

The methods used to generate the trial set of candidate plan variations at each

iteration of the goal planner influence the goal planner's operation in several ways. They determine the path which the goal planner takes through the search space. Therefore, they determine the areas of the search space which are reachable and how hard they are to reach. The goal planner is unlikely to reach the optimum plan if it must traverse unfavorable parts of the search space to get there. As the search is currently designed, the computational resources required for each iteration increase linearly with the size of the trial set. A small trial set is desirable to allow a large number of iterations in a given amount of planning time, though a large trial set has the advantage of thoroughly covering the neighborhood of the working plan at each iteration.

The trial set in our implementation consists of candidate plans produced by four types of variations of the working plan. Those variations are created by:

(1) adding goals to the plan,
(2) removing goals from the plan,
(3) moving a goal from one place in the plan to another, or
(4) using a different activity planner for a goal in the plan (discussed further below).

Moving a goal ((3) above) may also be achieved by removing a goal from one place on one iteration and then adding it in another on a succeeding iteration. Thus, removing and then adding a goal takes two iterations whereas moving it only takes one. Since moving a goal may make a large difference in a plan's value and resource use, it is desirable to allow moves as a basic part of the trial set.

One problem with iterative optimization schemes is that they can get caught in local maxima if the objective function is non-unimodal. A variant of simulated annealing is used to help overcome this problem [18]. In simulated annealing, there is a probability that the new solution generated by the optimization algorithm will be rejected if its utility does not represent an improvement over the current solution's utility. The probability of rejecting a solution of decreased utility increases as the search nears the end of its computational time allowance.

In this implementation, the probability of rejection of a solution has been included in the scoring functions of the goal planner so that a probabilistic selection need only be

made once. The scoring function is modified so that when the goal planner begins planning, it places little weight on value. More and more weight is placed on value as planning time runs short. Thereby, the probability of accepting a working plan of reduced value at a given iteration decreases as the time-available-to-plan is used up. This way, the search has a chance to escape local maxima when it has time, and it concentrates on finding the peak of the local maximum that it is probing when little time is left. Note that the implementation is not the pure form of simulated annealing in that the scoring functions are modified based on the *value* of the expected change to the plan, whereas in the pure simulated annealing approach the probability of rejection of a plan depends on the change in *utility* of the plan. Since value is tied very closely to utility, this approximation has largely the same effect as the pure simulated annealing approach.

The use of simulated annealing not only improves the performance of the goal planner, it also provides a mechanism for the planning manager to specify what type of planning the goal planner should do: if the manager just wants a plan refined, it gives the goal planner little time to plan and a high initial weight on value. If the circumstances warrant an exhaustive plan-space search, the manager allocates more time and specifies a low initial weight on value.

The scoring functions control the quality of planning in other ways as well. The scores are functions of resources used by a plan as well as of its value. Giving resource use a heavy weight can reduce the probability of violating resource constraints at a given level of the hierarchy. For example, if in a certain case the top level wishes to perform an unimportant task now and an important one later, it may command the unimportant one with a heavy weighting on resources to ensure that there are sufficient resources left at the end of the task to complete the more important task. The planner may also be instructed to produce plans which minimize uncertainty by heavily weighting the expected variations of value and resources, or it may be instructed simply to try for the best expected value of a plan by weighting only the expected value.

The steps performed at each iteration of the goal planner are:

1) Generate a trial set of variations from the plan.
2) Estimate the value and resource use of each trial plan.
3) Score each trial plan.
4) Probabilistically select one of the trial plans as the working plan for the next iteration.
5) Save the plan if it's the best one found so far.

4.6. ACTIVITY PLANNERS AND RESOURCE ESTIMATORS

Associated with each goal type is an activity planner and a resource estimator. The resource estimator must account for uncertainties in the models of the environment as well as in the models of the plant. Based on the models and their uncertainties, the estimator calculates the probability of success of the goal, and returns the expected value of the goal which is equal to the probability of success times the assigned value of the goal. The variance of the goal's expected value is calculated as well, from the variance of the probability of success of the goal. The accuracy of the resource estimators influences the effectiveness of the planning system; without accurate resource estimates, the planning system will make plans at the high levels which are initially either infeasible (not enough expected resources are allocated to complete the plan) or ineffective (resource expectations are overestimated, reducing the number of goals that can actually be accomplished). In both cases the system will have to replan as it discovers that it has more or less resources left than expected. If the resource estimates were perfect, the planner would never need to replan for those reasons.

The activity planners are the special-purpose agents at each level which translate the goals of the level into sets of goals for the inferior level. It is the activity planners which contain the knowledge of how to translate goals into the subgoals that are required to achieve those goals. An activity planner may draw upon any information sources necessary to decompose its activity into the goals of the next level. Generally, it will use extensive information about the environment to algorithmically compute the specifics of

the goal decomposition. For example, in the autonomous vehicle application, a straight-line survey activity planner plans a survey at the geographical location indicated by the goal's parameters, and decomposes this into survey lines. The survey lines are arranged in the survey area to cover the area efficiently and to avoid problems with any known obstacles or undesirable areas indicated in the mission map. The middle level has an activity planner for a survey line, which breaks down each survey line into a sequence of desired positions and velocities, and the bottom level's activity planner computes the maneuvers necessary to achieve the middle level's goals, and outputs goals for the control system.

Several versions of the same goal type, and therefore several versions of each activity planner, should exist for any goal which might be accomplished in any of several ways with different levels of resource consumption. The goal planner will then try different versions of each goal as it generates its trial set (see plan variation type 4 above), giving it a better chance of meeting the plan's requirements, since it will be able to tailor the resource use of each goal according to its context in the mission. For example, different transit activity planners might trade off time, fuel, risk, and navigational certainty differently.

5. MANAGER COORDINATION

The techniques used by the management functions to coordinate planning within and across levels of a hierarchical system have an enormous impact on the behavior of that system. Numerous coordination techniques have been developed for hierarchical systems which solve complex, static problems [19]. However, the planning problems which must be solved by autonomous systems are seldom static; the changing environment in which the autonomous system operates combined with the uncertainty inherent in the information about that environment place great importance on the real-time performance of the planning system.

In order to apply established coordination techniques to the dynamic, uncertain planning problem, the planning problem has been posed as a static problem that is frozen in the sense that the initial vehicle and environment state from which the plan will evolve and the models used to predict future states are assumed fixed at the moment at which the autonomous system begins to solve the planning problem. The uncertainty in the information upon which the solution is based is taken into account in that the objective of the search is to maximize the *expected value* of the mission given the available information, as described in Chapter 4. With the problem thus posed, a substantial amount of time spent in the search for an optimal solution may degrade the solution, because the information upon which the solution is based is aged by the amount of time spent searching. Therefore, the approach to coordination of the solution must both control the computational resources (time) spent in the planning and management processes and coordinate the solution to the planning problem in the traditional static sense, as discussed in Chapter 3.

The coordination functions therefore have been separated into *static coordination functions* which perform roles similar to those in the traditional hierarchical coordination

methods and *regulatory coordination functions* which control the dynamics of the process of searching for a solution to the problem. This search process occurs across the levels of the hierarchy as a complete plan is developed at all levels. The static functions reside in the temporal hierarchy, whereas the regulatory functions reside in the management hierarchy as defined in Section 3.3 and illustrated in Figure 1.

The static coordination functions coordinate the lower-level optimization problems of the temporal hierarchy by controlling the resources allocated to the plans at each level and by specifying the cost of over-running or the value of conserving each resource. Clearly, the resources available to the top level plan are fixed by the physical resource limits of the autonomous system, whereas at lower levels the resource allocations are set by the level above and may be traded off at the level above as needed with resources required for another part of the plan. The problem of an infeasible plan at a low level may be corrected by the level above by asking the lower level to accomplish fewer goals or by allocating more resources to it. When an infeasibility is detected at the top level, the only options are to compromise a goal or to create a more efficient plan.

The regulatory coordination functions also represent a form of resource coordination in the sense that they control the resources used by the planning system itself. They differ somewhat from the static coordination functions in that the computational resources allocated to planning across the levels must sum to the total computational resources available, whereas the vehicle resources required for executing the plans created at the various levels are simply different representations of the same quantity of resources. Thus, the *sum* of the three levels' use of computer time must be less than or equal to the actual computer time available, whereas the use of fuel over the span of the mission at *each level* of the hierarchy must be less than or equal to the total fuel available for the mission. Each set of constraints may be expressed as follows:

$$\sum_{j=1}^{N} res_i^j \leq b_i$$

where N is the number of subproblems being coordinated, res_i^j is the amount of resource i used by subproblem j, and b_i is the amount of resource i available to the overall problem. For each level of the temporal hierarchy, the set of subproblems is the set of N goals being planned by the level and the b_i is the allocation of consumable resource i to the level. In the management hierarchy, the subproblems under consideration are the levels of the temporal hierarchy, and b_i represents the total amount of computational resource i available to the autonomous system.

5.1. COORDINATION CONTROLS

The management functions of the management hierarchy coordinate the planning process through several mechanisms. They specify the computational resources to be allocated to each planning problem. They control the type of planning, from small refinements of a plan to complete overhauls of it, by allocating the computational resources and by specifying the initial annealing temperature of each planning job. They control the probabilistic character of the goal planner's search by specifying the character of its scoring functions, and they control the dynamic character of the entire temporal hierarchy by specifying the planning horizon at each level. Finally, the management functions of the management hierarchy must execute their solutions to the optimization of the planning process by determining when to replan and which levels of the hierarchy require planning or replanning.

The objective of the temporal hierarchy management functions is to maximize the objective function of the overall static planning problem such that it meets the resource constraints of that problem. The management functions of the temporal hierarchy coordinate the optimization of plans by setting resource prices, the cost or value of over- or under-running their budgets relative to the value of accomplishing goals.

5.1.1 Resource Reserve and Uncertainty

The level of reserve of a resource is equal to the total allocable amount of the resource minus the amount expected to be used during execution of the plan. Here, expectation is used in the statistical sense; the amount of a resource expected to be used during the execution of a plan is the statistical expected value of the random variable representing the amount of resource use, given the possible current states of the system and the environment:

$$E(res) = \sum_{\text{all S}} [(E(res) \mid S) \cdot P(S)]$$

where the S are all possible states of the system and environment, P(S) is the probability that S is the actual current state, and res is the amount of a particular resource used during execution of the plan.

Assuming that the information upon which the plan is based is not perfect, i.e., that the precise state of the vehicle and environment is not known, execution of the plan with no allocated reserve of a given resource would probably exhaust that resource about half of the time, depending on the actual shape of the probability distribution for resource use (see Figure 8).

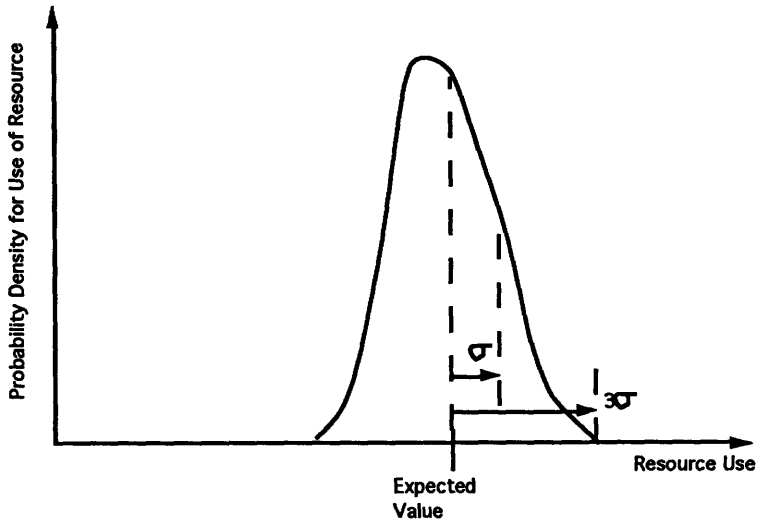

Figure 8: Probability Distribution of Resource Use During Plan Execution

If a resource reserve of one standard deviation of the resource use is allocated, the probability of exhausting the resource becomes more reasonably small, and a resource reserve of three times the standard deviation is generally quite safe. As the mission evolves and more information about the state of the vehicle and of the environment is gathered, the standard deviation of the resource use decreases, and the resource reserve may be decreased while maintaining the same level of confidence that the resource will not be exhausted.

Ideally, one would like to allocate the amount of resource reserve which maximizes the expected value of the mission. A small resource reserve makes completion of the final phases of the mission uncertain, since some critical resource may have been exhausted by the time that part of the mission is reached. A resource reserve which is overly conservative tends to limit the usefulness of the mission, since the reserve takes resources away from the pool of resources which would in other cases be used to achieve mission goals. By using the probability distribution functions for value and resource use of individual parts of the plan, it would be possible to precisely determine the optimum resource reserve to achieve the best possible expected plan value.

In practice, the precise probability distribution functions for resource use of the plan conditioned on all possible future states is seldom known, because the calculation of these functions would require extremely accurate models of the evolution of the state of the vehicle and environment and would consume significant computational resources. In many cases, the mission situation could change enough to make the functions inaccurate by the time they were calculated. In addition, development of the vehicle and environment models would require vast engineering resources which could be better spent advancing the state of the art in robotics technologies, reducing the uncertainties involved in the autonomous system, improving its capabilities and expanding its resources.

Despite the difficulties involved in precisely evaluating the optimal reserve, the use of a few approximations yields a much more readily solved problem. By assuming the probability distribution function to be Gaussian, the calculations involved in arriving at the distribution are orders of magnitude simpler, yet they still yield an accurate or at least sufficient representation of the real distributions. The calculated resource reserve then accounts for uncertainty, and equally important, it is available quickly and without consuming inordinate amounts of computational resources. As the mission progresses, the uncertainties in the information about the mission will decrease, and the resource reserve may be adjusted accordingly.

Tasks which add small amounts of value to the mission but which are not critical to the mission's success naturally tend to be planned late in the mission as uncertainty decreases and resource reserve may become available for use. Tasks which are vital to the mission are normally planned to occur early on, so that if budgets are over-run the important parts of the mission can still be tackled. The use of a resource reserve calculated based on the uncertainty in resource use of the plan results in this logical planning behavior.

5.1.2 Price Coordination

The resource reserve is a tool that can be used by each planning level of the hierarchy to control the tradeoff between uncertainty of the future and the resource requirements of the part of the plan which is currently executing. Resource price coordination is used to correlate the resource reserves at different levels of the hierarchy. A superior level of the planning hierarchy communicates to its inferiors the cost of dipping into the superior's resource reserve and the value of adding to the resource reserve. The inferiors then incorporate this cost into the optimizations of their plans. In this way, the lower level gains information from the upper level about the relative importance of the lower level plan to the overall mission plan. The lower level may decide to use the extra resources if the price of the resources is low enough, or it may modify its plan to use less of the resources if the price on those resources is high.

The resource prices must be accurate enough to ensure the viability of this coordination method. If the prices were inaccurate, the lower levels' views of the global optimization problem would be distorted. The incorrect view of the global problem would lead the lower level planners to create plans which were inconsistent with the overall mission objectives, and then the superior level would need to modify the prices and try the plan again. In a real-time system, iterations across planning levels are expensive since they expend some of the mission's finite computational resources. Incorrect prices, however, are potentially disastrous since they cannot be readjusted for those parts of the plan which have already been executed.

The prices must therefore reflect the value to the overall mission to be gained or lost by the addition or removal of available resources. A precise approach to the calculation of the prices would be to optimize the mission with a wide range of total resource allocations, and to calculate the value of the best mission found in each part of the allocation range. The resource prices would then be a function of the amount of resource being conserved or overrun, and would very accurately reflect the importance of

the resources in the context of the overall mission.

The problem with this precise approach to calculating the resource prices is that it would require an extensive amount of computational resources. For most real-time applications, a planning system which calculates a resource price function based on enumeration of the range of probable resource allocations will perform worse than a planning system which merely approximates this enumeration because the extra computational resources required for the exhaustive method of determining prices cost the mission more than do the small inaccuracies in the approximate method.

A practical price-setting method for systems which are limited by real-time constraints is to consider only small changes in resource allocations and to linearize the prices about those small changes. The price of over-running a given resource may be found by determining the cost of dropping from the plan the goal with the greatest yield of the resource for its value, and then dividing that cost by the amount of resource gained in the process. Overruns in the resource would cause the given goal to be dropped from the plan, costing its value, hence the approximation. The value of adding additional resource may be calculated by determining the first increase in value to be had by increasing the resource. Going under budget on the resource would in most cases be worth less value than overspending would cost, however, since there should be no goal which when added to the plan could use the resource and return better value than the goals currently in the plan. Since the value of additional resource is less than the cost of the resource shortage which was calculated, the price of the resource at the current resource level should be somewhat smaller than that for dropping a goal and greater than that for adding one. The price at the current level is taken to be an interpolation of the two values, or in cases where computational power is very limited, it may be taken to be a large fraction of the resource shortage cost.

This approximation to the resource price works well for small changes in resources. Large changes in resources, however, are out of range of the linear

approximation. In the event that a sub-problem requires a large change in its resource allocations, it will likely fail to produce a solution that is optimal in the global sense. This will be reflected in the fact that its solution's value and resource use will not match the expectation for value and resource use determined by the superior level, and the superior level will need to be invoked to replan the global problem given the new information.

5.1.3 Scoring Functions

The scoring functions determine the relative depth and breadth of the plan optimization search performed by the goal planner. They calculate the score of each of the trial plans generated in each iteration of the goal planner based on the plan's value and resource use. The goal planner uses the scores to probabilistically select the next plan in its search, so the scores reflect the importance of value and resource use.

By specifying the scoring functions, the management functions control the nature of the search. The importance of resource use as reflected in the score of a plan should be related to the resource prices at the level, however they need not correlate exactly. The goal of the search is to transit the search space in such a way that a near-optimal solution is found, so the scoring functions should be designed to guide the search toward optimal solutions rather than solely to reflect the worth of a particular solution. The actual value of each selected solution is evaluated by the objective function of the planning level that determines which of the plans found in the search would be the best to execute.

Therefore, the scoring functions provide the manager with a control to ensure that the searches find results in the amount of time available. If the manager needs a solution quickly, it should employ scoring functions which differentiate heavily between solutions with slightly different utilities, so that the iteration is more likely to lead the search up the nearest local optimum rather than considering branches which may or may not later result in more global optima. If, on the other hand, there is ample time to find a solution, the

manager should specify scoring functions which weight the change in the plan's utility less, so that the search covers a broader area of the search space and has the opportunity to escape local optima.

5.1.4 Initial Annealing Temperature

The annealing temperature controls the plan-space search in a manner similar to the way the scoring functions do. In fact, the annealing temperature in this implementation modifies the scoring functions to place less importance on the value of the plan's value initially, and then as the goal planner uses up its searching time the importance of the value of value increases, so that the search tends to initially escape local optima and then gradually find more global optima. This implementation of simulated annealing differs from the rigorous definition of simulated annealing only slightly, as discussed in Section 4.5.

The planning manager specifies the initial annealing temperature in order to control the evolution of the importance weights used by the scoring functions as the planning time is spent. Specifying a high initial temperature (one that places little weight on value) causes the search to start out examining a broad range of alternatives, and then narrow in on a local optimum when the temperature cools near the end of the planning time. Specifying a low initial temperature causes the search to concentrate on finding a local maximum right away, without broadly examining the alternatives. A low initial temperature, and similarly a set of scoring functions which differentiates highly between plans of slightly different expected value, should be used to optimize a plan which is already very likely to be near-optimal, and a high initial temperature with a set of scoring functions that differentiate only slightly between plans of similar expected value should be used when the search needs to examine different possibilities without the aid of a good starting point in the search space.

5.1.5 Planning Horizon and Uncertainty

One major difference between typical problem-solving decompositions and real-time planning problem decompositions is that much of the detail in the real-time planning problem is unknown for the parts of the problem which occur in the future. Usually, the parts of the problem in the near future are better defined than those in the distant future, since information about the future degrades due to any number of uncertainties the further into the future the information is extrapolated. The greater the uncertainty in the details of a part of a problem, the less detail can be meaningfully planned.

Therefore, the level manager must control the planning horizon of its level to ensure that the details of the level are relatively certain. If the planning horizon extends too far into the future, the parts of the plans at the far reaches of the horizon will be based on information which is likely to change, requiring replanning and effectively representing a waste of computational resources applied in developing the distant future parts of the original plan. Alternatively, if the horizon is too short, the plans created will not be as useful as they might have been had they taken into account further extrapolations of the future. The planning manager needs to balance the cost in computational resources of evaluating the plan at a given horizon versus the value to be gained in considering the events which will happen in the future, given the certainty with which the projected future will happen.

5.1.6 Planning Horizon and Hierarchy Dynamics

The planning horizon also affects the dynamics of the interactions among the levels of the planning hierarchy. A horizon which is short will require frequent replanning, since a new plan must be generated as the old plan is executed. A horizon which is long will not result in frequent replanning due to completion of plans; however, if the horizon is too long, replanning may be required before the plan is completed because the plan may become obsolete due to uncertainties in the mission environment

that unfold during plan execution.

If the planner's computational requirements approach the limits of the total available computational resources, it will become important for the hierarchy to replan only as often as necessary. A planning horizon which is very long will consume large amounts of computer time, since the search space grows exponentially with the length of the plan. On the other hand, a horizon which is very short requires large amounts of computational resources since it needs to be replanned often. Any replanning at a high level will require replanning at the inferior levels if any of the goals which those levels have planned out in detail are changed by the high level.

It is possible for the hierarchy to replan continually if the horizon is either too long or too short. If the horizon at a level is too short, the system may spend all its time replanning plans which are executed as soon as they are planned. Similarly, if the horizon is too long, the planner may waste too much time planning for the far future to be able to keep up with the pace of events in real time. There is an optimum planning horizon somewhere between these two extremes which will allow the planner to perform effectively with an efficient use of computational resources. The dynamics of the hierarchy need to be modeled in order to determine that optimum planning horizon for any given application of the hierarchy.

5.2. CONTROL OF PLANNING QUALITY

The manager chooses the planning coordination variables to maximize the utility that can be achieved in the current situation. Making this choice requires a model of the goal planner's performance as a function of those coordination variables.

The top-level planner's performance is examined in two different situations below. The first situation is one in which the initial plan given to the planner is empty: the goal planner must create the entire plan from scratch. In addition, for the first situation there

are hard constraints on fuel and time. That is, exceeding those constraints reduces the value of the plan, since no mission goals are completed beyond the point in the plan at which a critical resource is exhausted. Situation two considers the case in which the hard constraints are not binding at the optimum. As in situation one, the initial plan is empty.

In each situation, two approaches to coordination are examined, one of which has a high initial annealing fraction, and one of which uses an initial annealing fraction of 0. The initial annealing fraction is the fraction of the scoring functions' weight on plan value that is used in the first iteration of the goal planner. The annealing fraction increases linearly with time so that it reaches unity at the completion of planning. Thus, the score of each plan perturbation is calculated as follows:

$$\text{score} = w_v \cdot a \cdot E(v) + \sum_{i=0}^{N} w_i \cdot E(r_i)$$

where w_v is the value placed by the scoring function on utility before the annealing temperature is taken into account, E(v) is the expected value of the plan, w_i is the value placed by the scoring function on resource i, and $E(r_i)$ is the use of resource i expected during execution of the plan. The factor a is the annealing fraction, which varies with time in this manner:

$$a = a_0 + (1 - a_0) \cdot \left(\frac{t_f - t}{t_f - t_0}\right)$$

where a_0 is the initial annealing fraction, t_0 is the time at which planning was begun, t_f is the time at which planning is to end, and t is the current time.

In situation one, below, the objective function is nearly unimodal along the trajectory of plans that is followed by the planner. In this case, the best strategy is to select perturbations to the plan which maximize the objective function.

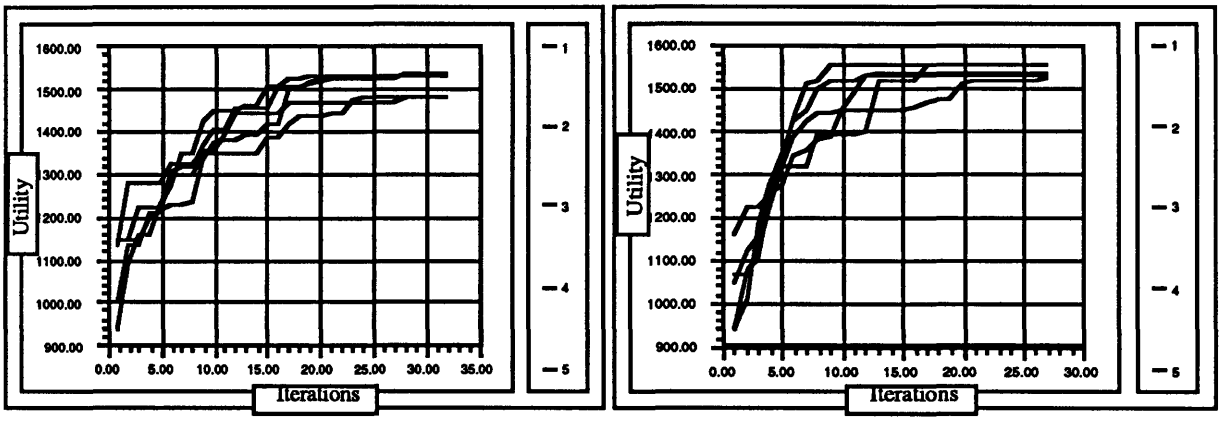

Figure 9: Five plan utility histories with an empty initial plan. On the left, the planner used an initial annealing fraction of 0, while on the right, the initial annealing fraction was .9.

Figure 9 shows the performance of the planner in situation one at the two different settings of the initial annealing fraction. The utility of the plan is shown as a function of the number of iterations of the goal planner. Each case took 40 seconds of planning time and represents planning out a mission consisting of ten waypoint goals. In all cases, the planner searched the plan space with respect to (a) the set of goals to be included in the plan, (b) the ordering of the set of goals, and (c) the speed at which the vehicle traveled to each of the waypoints.

As might be expected, the planner which used the higher annealing fraction climbed to the optimum more quickly and reliably than did the planner operating with the low annealing fraction. In this nearly unimodal objective function, the value of the mission increases at most of the perturbation steps in the search from the empty initial plan to the optimal one. For this reason, the planner which places a high weight on value (the one with the high annealing fraction) is more likely to choose the correct perturbation at the majority of steps, so it reaches an optimum more quickly than does the other.

When the optimal solution is well within the resource constraints, the problem effectively becomes unconstrained, so the probability of failure due to running out of constrained resources factored into each goal will be 0% and have no effect on the maximization problem. This situation is shown below in Figure 10.

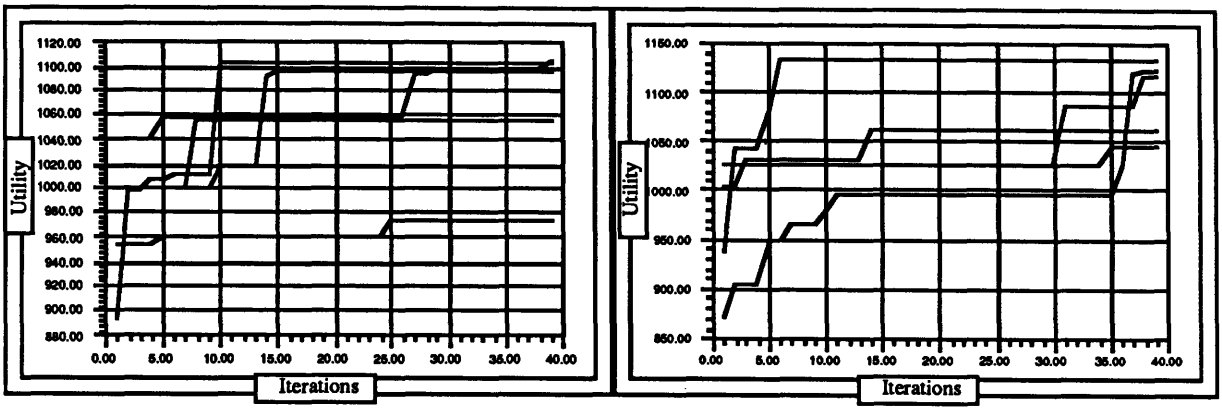

Figure 10: Effect of initial annealing fraction on the unconstrained planning problem. On the left, an initial annealing fraction of 0 was used; on the right, the initial annealing fraction was 0.9.

Given that in this case the penalty functions don't influence the plan utility, the effect of the initial annealing temperature on the planner's performance is seen to be much greater. In the left-hand graph, a low annealing temperature was used, helping the planner to escape any local optima in an attempt to reach the global optimum. In the right-hand graph which shows the results achieved using a high initial annealing fraction, the progress of the planner toward the optimum is not so rapid, particularly at first. Note that although the planner with the low initial annealing fraction escapes local optima initially, it may still become trapped in local maxima if it happens to reach one from which large changes in the plan are required to reach the global optimum, as seen in the one trial which never achieves utility greater than 1000.

From these results, it can be seen that simulated annealing has the greatest effect when the resource constraints aren't active. This is because the resource constraints limit the scope of the search to regions of the search space which are likely to prove fruitful much more effectively than simulated annealing does. Comparison of the cases that are actively constrained by the resource allocations with those which aren't shows that the planner performs much better when it has constraints to guide it than it does when simulated annealing is its only guide. Figure 9 shows much more rapid and consistent progress than does Figure 10. Another interpretation of this result is that the search

performs best when the scoring functions used to select the next step at each iteration are closely related to the actual utility of the plan. When the hard constraints of fuel and time come into play, the value of the plan drops off rapidly as it violates those constraints. A similar drop in the utility of the plan is realized due to the drop in value when the constraints are violated, and the scoring function behaves similarly to the utility function. Without the hard constraints, the search algorithm does not have such a nice, sharp boundary to follow through search space.

This result suggests that the best use of the goal planner during time-critical replanning is to give it a resource allocation close to the expected resource use of the optimal plan. The resource use expected of the plan at the superior level is probably close to the optimal plan's expected resource use and may be used as a guide to set the lower-level allocations. The annealing fraction should be set according to the time available to plan and the expected change required in the initial plan; an initial plan known to be close to the optimum (because it was just recently created, for instance) may be planned with a high annealing fraction in a short space of time. More significant changes to the plan may require a lower annealing fraction at first, particularly if the resource requirements of the plan aren't well known. Another strategy might be to replan several times in a row with different resource allocations when the amount of resources required isn't well known, and then to choose the best result.

6. IMPLEMENTATION

The mission planning and control software is implemented in the C language on a Macintosh computer. It is being used to plan missions for and to implement the control of a simulated vehicle in a simulated environment. The simulations also run on the Macintosh and are interfaced to each other and to the planning and control software through NetSim, a software package developed at Draper Laboratory that is useful for many types of closed-loop simulation problems.

6.1. THE IMPLEMENTATION ENVIRONMENT

The mission planning and control software interacts with a vehicle dynamics simulation, an environment simulation, a simulation of the vehicle sensors, and a simulation of the vehicle's pattern-recognition software. Each of these simulations provide information to the mission software comparable to the information which would be provided in a real submarine for tests performed at sea.

6.1.1 Vehicle Dynamics Simulation

The vehicle simulation is based on a model of the Sea Squirt vehicle [4]. That model includes hydrodynamic forces, inertial forces, and actuator thrusts in six degrees of freedom. The model accepts actuator command information from the control software in order to compute the actuator thrusts. Information about currents and obstacles that is provided by the environment simulation is used to calculate hydrodynamic forces, and the vehicle simulation's internal information about the vehicle's configuration (i.e., weight, displacement, drag coefficients, and moments of inertia) is used to calculate the response to these forces in terms of the trajectory of the vehicle's state through 6-dimensional state

space. The resulting trajectory information aids the sensor simulation in determining the information collected by the sensors.

6.1.2 Control System Simulation

The control system that is used to provide actuator commands to the vehicle simulation is the sliding mode controller that is currently implemented onboard the Sea Squirt vehicle. The C code for the controller was modified from the version used on the vehicle to accept the inputs and outputs provided by the NetSim simulation environment rather than the actual hardware I/O used on the vehicle. The modified code was recompiled to run on the Macintosh under NetSim.

The control system consists of two independent controllers which treat the vehicle's heading, depth, and speed as if they were not coupled. For the Sea Squirt this assumption is reasonable, and the control system works well as designed. Closed-loop control of the vehicle's depth and heading is adequate, but speed is controlled open-loop. On the actual vehicle, there is no reliable speed sensor, so the speed command to the controller is simply translated directly into a thrust (power) command. The designer of the planning system must therefore account for the poor control of the vehicle's speed by ensuring that the lowest-level activity planners carefully monitor the plan's progress. The set of goals should be decomposed in a manner that does not require too much speed control accuracy; the lowest-level activity planner should be tolerant of speed variations.

6.1.3 Environment Simulation

The environment simulation provides information to the other simulations about the obstacles, currents, and terrain features it is simulating. These features can be changed on-line by the user so that the effects of different mission scenarios may be played out fully.

Obstacles are represented as either spherical or cylindrical objects. They may be

positioned, scaled, and oriented by the user in any part of the environment.

Currents are represented as static flow fields created by pseudo-sources, sinks, and vortices. They also may be modified by the user.

The terrain is presently represented by a function which returns the depth of the water column. The profiles may not be modified on-line, though they provide a surface detailed enough for the purposes of these tests.

6.1.4 Sensor Simulation

The sensor simulation models use information from the vehicle and environment models to produce sensor outputs. Currently, the simulation includes models for vehicle attitude sensors, wide-beam obstacle sonar, and altitude sonar.

6.1.4.1. Sonar

Three forward-looking sonar beams comprise the obstacle sonar. A downward-looking sonar beam measures the altitude of the vehicle. Taking into account the position and orientation of the vehicle, the forward-looking sonar model determines the conical regions of space covered by each sonar beam, and returns for each beam either the range to the nearest obstacle or an indication that no obstacles are in the cone associated with the beam. The model does not attempt to simulate ghost images, false alarms, or other typical sonar problems, although there is a minimum range below which it will not detect obstacles.

The altitude sonar uses the position of the vehicle and the depth of the water column at the vehicle's position to return the distance to the ocean floor. Again, the detailed errors of a sonar system are modeled as a single error term added to the true altitude. The simulation is intended to test the mission planning and decision-making software's abilities, not the abilities of a particular set of sensors and any particular sensor fusion algorithms.

6.1.4.2. Depth, Pitch, and Roll

The depth sensor is modeled as a simple pressure transducer. The accuracies of depth sensors are high, and their small deviations from ideal are not significant enough, relative to the accuracies required for our purposes, to warrant modeling. Therefore, the depth sensor model merely reproduces the actual depth of the vehicle.

Similarly, the pitch and roll sensors are sufficiently accurate for mission planning purposes, and their dynamics are considerably faster than the dynamics of the vehicle, so they are modeled as ideal sensors. A typical attitude sensor consisting of a bubble in a viscous fluid responds to disturbances in the vehicle's orientation much more quickly and accurately than is required for the purposes of this test. Since the errors of an attitude sensor are insignificant at the level of the mission software, the actual simulated pitch and roll of the vehicle are taken directly by the mission planner as sensor inputs.

6.1.5 Obstacle Mapping Simulation

This architecture requires information processing algorithms to provide the planners at each level with information of a scope and detail commensurate with that level. For the purposes of this AUV example, the information required is primarily geographical in nature so it is represented in maps which are stored in the computer's memory as lists of obstacles. The information provided by the maps consist of obstacle locations and sizes. This information allows the planners to plan trajectories and paths around the obstacles.

An assumption that is inherent to the simulation is that the vehicle will recognize the position and size of any obstacles seen by the sonar. This is unlikely to be achieved in a real application, but it provides a basis for testing the decision-making architecture without the complications of developing the complete information fusion routines.

Once an obstacle is sighted, its properties are added to a list of sightings which constitute the vehicle's map of the environment. The activity planners have access to this list as obstacles are added to it. In the current implementation, the initial list of obstacles

is empty, and the only change which can occur to the list is to add an obstacle to it. A more complete information system would allow an initial model of the environment to be updated as the mission progresses, in which case obstacles might exist in the initial model which weren't in the actual environment or environment simulation and should then be removed upon detection of their absence. In addition, it would be desirable to provide maps of a different level of detail at the different levels, so that the top level planners could evaluate their plans in less detail than the bottom levels. Such an implementation of the mapping system, relying on quadtrees, is under development.

6.2. PLANNING MANAGER

The Planning Manager is responsible for the coordination of the three levels of planning, as described in section 4.4. The primary tasks of the planning manager are to direct the planners to produce feasible plans, to monitor plan execution for unexpected problems and opportunities, and to direct replanning as required during plan execution.

6.2.1 Directing Planning

The Director function within the planning manager directs planning and replanning. It maintains the plans at each level in five different structures: *new_goals*, *initial_goals*, *working_plan*, *best_plan*, and *estimate*. New_goals is the command handed down from the level above in its last replan; the portion of new_goals which is to be replanned is copied to initial_goals each time the plan at a level is replanned, and then the replanning process uses the goals in initial_goals to create a new plan. Thus, initial_goals contains the goals which were optimized by the goal planner to create the best_plan at the level. Working_plan contains the plan which was selected in the last iteration of the goal planner. It is used to specify the trial plan used by the goal planner, and it is also used to store the state of the goal planner so that control may be returned to the rest of the system

when the process of goal planning is interrupted. This allows the goal planner to be restarted from where it left off the next time the director function is executed.

Estimate contains the best estimate of the degree to which the plan has been executed, i.e., its *execution status*. It also contains the best estimate of the results which will be achieved by executing the plan. When the goal planner finishes adding new goals to the plan, the quantities of value and resource use for those goals stored in the estimate are simply the estimates made by the goal's estimation routines. Then, as the plan is refined at lower levels and eventually executed, the estimate at each level is updated to incorporate the estimates of the levels below it. When replanning occurs, the estimates of resource use and utility of the plan are taken into account in the resource estimation routines. In this way, if execution of a goal has been attempted and failed, the planner will know it when it replans. If the system didn't take into account knowledge gained during plan execution, it would be unable to meaningfully replan in situations that it did not model.

The planning director is invoked at regular intervals by the simulation software, just as it would be by the operating system of an autonomous system. The director begins by updating the resource estimates and execution status estimates at each level of the plan, and from these it is able to determine the replanning urgencies at each level. It uses the replanning urgencies at each level to decide which level needs to be planned first, and then it calls upon Goal_Select (discussed in Section 6.2.1.1) to choose the goals to plan and the resources to allocate. Next, it calls the Goal Planner to create a plan from the goals. Finally, it uses the appropriate activity planner for each goal of the plan to create subgoals, and modifies the plan in the next level's goal buffer to incorporate these new subgoals.

During this process of specifying the subgoals from the goals, the plan manager maintains a record of which subgoals originated from which goals. While monitoring plan execution, the manager will use this record to update the estimates at higher levels

based on those at the lower levels so that the plans remain consistent across the levels.

6.2.1.1. Goal_Select

Goal_Select is used by the management hierarchy to allocate the resources and choose the goals for the goal planner to optimize. These allocation and selection functions have a significant influence on the problem-solving dynamics of the real-time hierarchy. The length of time it takes the goal planner to search for a near-optimal plan depends on the size of the search space; the size of the search space increases exponentially with the number of goals being planned, so the use of planning time is tied closely to the number of goals planned at each level. Allocation of too much or too little of any resource can result in an inefficient or infeasible overall plan, necessitating replanning.

In selecting the set of goals to send to the goal planner at a given level of the planning hierarchy, Goal_Select compares the current plan at that level with the goals commanded by the higher level. Three selection criteria are considered: 1) The difference between the expectation of utility calculated when the plan was originally created and the most recent prediction of the plan's utility based on the current state and progress of plan execution, 2) Which, if any goals commanded by the superior level have changed since the last replanning cycle on this level, and 3) The difference between the remaining time span of the current plan and the desired temporal horizon for plans at that level. Events, failures, performance and model changes that affect the prediction of the plan's utility are taken into account in the first criterion. The second and third criteria ensure that the plans of the level are consistent with the superior's plans and that the plans of the level are of the proper temporal scope, respectively.

Goal_Select allocates resources according to a deterministic heuristic. The heuristic classifies the type of goal planning problem according to the selection criteria above, and allocates resources according to the class of the problem. If the difference between the resource use expected when the current plan was created and the actual

resource use of the plan is large, the heuristic assumes that something has gone wrong with the plan and that a thorough replanning job may be needed. It directs the goal planner to replan with a low initial annealing fraction, a slightly higher resource allocation than is expected to be needed, and a large amount of time to replan. If, on the other hand, the old plan were performing well but were running out of plan horizon (i.e., nearing completion), the planning is expected to proceed normally. In this case, a smaller resource allocation is prescribed than in the previous case, and a high initial annealing fraction is used. In the final situation, when the goals commanded to the level have changed recently, the command is assumed to be based on resource estimates which reflect the current state of the environment and vehicle since it was created recently. In this situation, little time is allocated for planning, a high initial annealing fraction is used, and the resource allocation matches the expected resource use of the goals quite closely.

Note that the resource allocations given to the goal planner serve to guide the scoring functions in the search for the optimal plan. They are usually higher than the actual resources warranted by the level. The goal planner does not enforce any strict resource allocation on its plans. Rather, it selects the plan with the maximum utility encountered in its search as the best plan. The search is guided by resource prices as well as value. Thus, the utility of the plan takes into account the resource prices, which are designed to limit the resource use of the level.

Presently, the goal selection routine considers only the time horizon of the level in deciding how many goals to plan. The selection routine truncates the commanded goals to a sequence that is long enough to reach the time horizon of the level. The computational time allocated to planning is based on a heuristic that considers the size of the planning problem as well as the class of the planning problem determined above. If the new plan is expected to be short, little planning time will be required, whereas if it involves a long and complicated sequence of goals, more planning time will be required. This time-to-plan requirement is multiplied by a planning time factor determined in the

heuristic that allocates resources.

6.2.2 Plan Execution Monitoring

Plan execution is monitored by the Estimate Update function of the planning manager. This function operates once at the beginning of each planning cycle to incorporate the latest knowledge of the progress of plan execution into the estimates of plan utility and resource use at each level. Each goal's expected resource use is equal to the sum of the resource uses of its subgoals. Progress toward completion of each goal is determined by the progress made toward completing its subgoals.

The actual calculation of the estimates starts at the bottom levels of the hierarchy and works up to the top. The status of execution is known precisely at the bottom level, where discrete control commands are issued and their completion can be measured directly from the results of simple sensor processing. The use of resources and completion of goals at the bottom level is used to determine the resource use and execution status of goals at the next level up, since these quantities at the superior level are the sum of quantities at the inferior level. The process continues its way up the hierarchy, until the estimates at all levels are current.

After updating the current execution state of the plans at each level, the estimation functions are called to project the current state forward through the plan in order to predict the effects of the current state on the future of the mission. After completing this step, the plan estimates at each level reflect the expected value and expected resource use through the end of the time horizon of the plan at that level.

The update function must also maintain the *execution pointer* at each level. The execution pointer points to the first goal of the plan which is currently being executed but has not been completed. Every replan must take into account goals starting at the execution pointer and extending the length of the planning horizon from that point, allowing goals which are currently executing but not yet completed to be replanned.

Since subgoals may be rearranged at the inferior level, execution at the superior level may not proceed directly from one goal to the next. For example, Figure 11 shows an example of two goals originally planned at a superior level. Each is split into several subgoals by that level's activity planner.

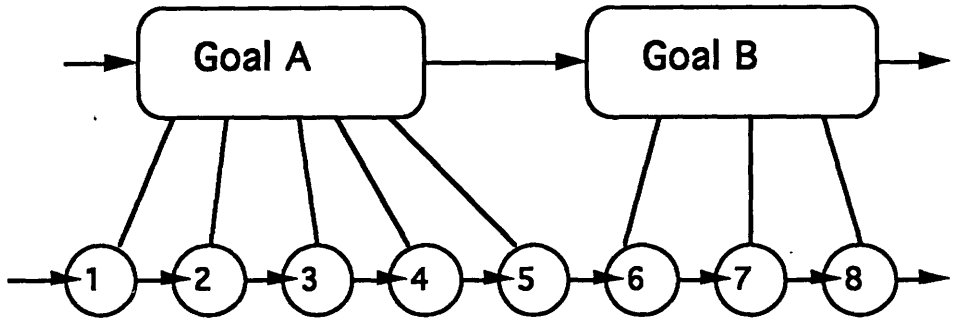

Figure 11: Two Plan Nodes with Subgoals

The subgoals may be rearranged by the planning at the inferior level, resulting in the situation in Figure 12. When execution reaches subgoal 6 at the inferior level, execution of goal B has begun even though goal A hasn't yet been completed.

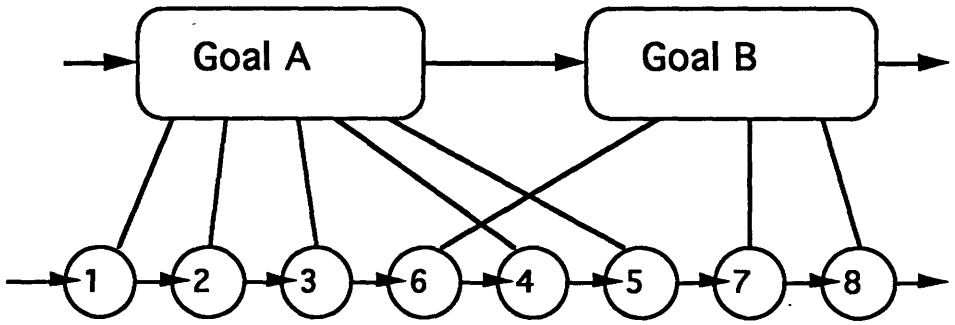

Figure 12: Subgoals have been reordered

Through this type of reordering and modification of the plan as its details are filled in by the lower levels, the concept of execution of a goal becomes complicated. Therefore, the following definition of execution is used: A goal is *executing* when at least one of its subgoals has been executing but all subgoals have not completed. It is *completed* when all of its subgoals have completed. At the bottom level, a goal is executing when it is

being carried out by the control system, and it is completed after it has been achieved by the control system.

6.2.3 Plan Decomposition using the Activity Planners

The planning director is responsible for decomposing the plan into a command to the next level down in the hierarchy. It invokes the activity planner for each goal of the level's plan, and the activity planner then returns a set of subgoals, linked together in a segment of plan, which are designed to accomplish the goal. It is the task of the planning director to connect the plan segments created by the activity planners together into a plan of the same form as the level's original plan (see Figure 13).

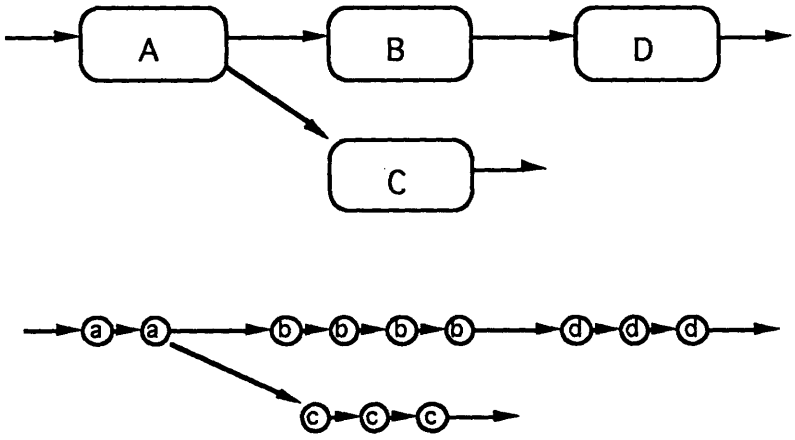

Figure 13: The original plan (top) is decomposed into subgoals which are then arranged (bottom) in the same overall structure as the original plan.

6.3. GOAL PLANNER

The goal planner, as described in Section 4.5, is a probabilistically-directed optimization algorithm which employs a form of simulated annealing. The probabilistic nature of the algorithm together with simulated annealing help overcome the non-convexity of the planning problem. Convexity of the objective function cannot be

guaranteed for this problem because the precise nature of the plan space and the associated utility function surface are not known until the architecture is customized for a particular application. Therefore, the planning algorithm must be capable of dealing with non-convex problems.

The goal planner consists of several parts as shown in Figure 14. Its main loop maintains a best plan and a current plan, and performs a search through plan space by perturbing the current working plan. When the search leads to a plan which is better than those found before, the goal planner stores it as the best plan, and continues searching until allocated time-to-plan is up. On each iteration, the main loop invokes the trial-set function which creates a list of candidate perturbations to the current working plan (i.e., a list of trial plans), and evaluates the trial plans using the resource estimators which calculate the resource use and expected utility of the trial plans. Each trial plan is given a score by the scoring functions, according to its resource use and expected utility. Then, one of the trial plans is selected probabilistically according to its score, and that plan becomes the new working plan.

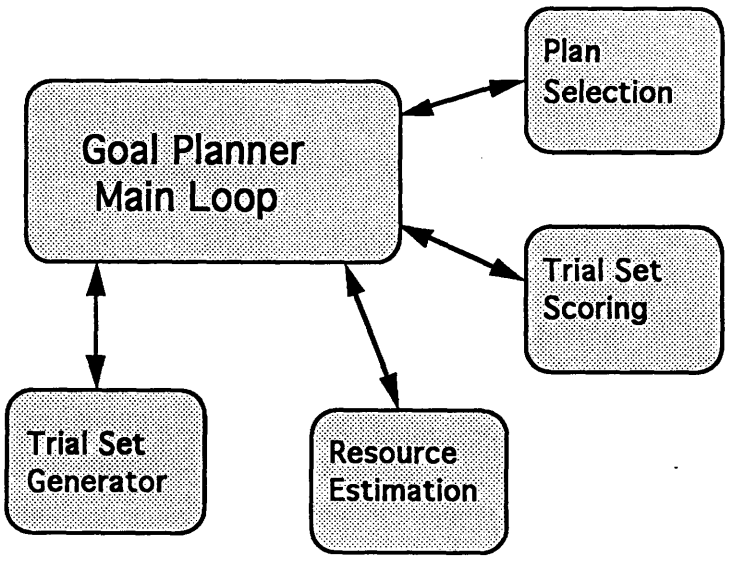

Figure 14: Goal Planner Composition

The scoring functions determine the direction in the search space that the

optimization routine takes at each step of the search. A single candidate plan is selected by adding the scores of all the trial plans and choosing a random number between 0 and the sum of the scores. The plans' scores are then added together again, and the plan whose score brings the total above the random number is selected as the next plan. Formally, the selected plan is the p^{th} plan of the trial set of plans, where p is:

$$p = \arg\min[n]$$
$$s.t. \sum_{j=1}^{n} sj > rand(\sum_{j=1}^{N} s_j)$$

where N is the total number of plans in the trial set, rand(x) is an instance of a uniformly-distributed random variable between 0 and x inclusive, and s_j is the score of trial plan j.

For this selection process to work, each score must be a positive number representing the worth of its plan. The score of the trial plan is computed as follows: the change in each resource, $\partial_{res\ i}$, is the difference between the use of resource i in the trial plan and its use in the current working plan. A weight w_i is calculated for each of the resources as:

$$w_i = C_0 + C_1 e^{(C_2 \cdot resnorm_i)}$$

where C_0, C_1, and C_2 are constants associated with the particular resource, and $resnorm_i$ is the normalized amount of resource i used by the current working plan (the same form is used to weight the plan value, in which case $resnorm_i$ is replaced by the normalized amount of utility of the current working plan). C_0, C_1, and C_2 are chosen such that the weight of a resource increases when the amount of that resource used by the working plan is near the allocated amount. Proper selection of these constants encourages the planner to search only in regions of the search space which are likely to be feasible and to produce value. $Resnorm_i$ is the following quantity:

$$resnorm_i = \frac{res_i}{alloc_i}$$

where res_i is the amount of resource i used by the current working plan, and $alloc_i$ is the

amount of that resource allocated for use. With the weights calculated as above, the score of the plan is computed as:

$$s_j = \sum_{i=1}^{M} w_i \cdot \partial_{res\ i}$$

where M is the number of consumable resources of the system.

6.4. GOAL SET

Since the capabilities of this vehicle consist primarily of navigation and data collection, the goal set has been designed to perform navigational tasks which support data collection. The top level consists of *transit* goals, and is also meant to readily support *survey* goals. Each of these goal types is decomposed into *waypoint* goals which the middle level can interpret. The middle level's waypoints are broken down further into *maneuver* goals and *travel* goals by the bottom level. The output of the bottom level is executed by the *plan spooler* which directs the vehicle's control system to follow the plan.

The top level activity planners use map data to plan feasible paths around any obstacles that have been detected to be present. The algorithm used to create an obstacle-free path performs a simple iteration. It starts with the path directly between the current position and the goal position, and checks to see if there are any obstacles along it. If so, it creates another goal slightly beyond the edge of the obstacle which is closest to the path, so that the path now consists of two shorter segments which avoid that one obstacle. The algorithm is then applied to each of the two segments, and so forth until there are no obstacles intersecting the path.

The middle level of activity planners perform the same operation as the top level. If it were in a system with a fully-developed information fusion system, the middle level would use more detailed maps than the top level does, and create the path more precisely. The bottom level uses this path data and creates plans which follow it, deviating from the

commanded path only to avoid obstacles which were not known at the time it was created.

The bottom level works with a set of models more detailed than those of the middle level. It breaks down the commanded series of waypoints into trajectory segments which the vehicle's control system is capable of following. It also ensures that these trajectory segments do not lead the vehicle into any obstacles by specifying a minimum turn radius larger than the obstacle when planning a trajectory around it.

The bottom level activity planners are capable of generating two types of trajectory commands to be output to the control system. When the vehicle is to cross large regions of space between waypoints, the activity planner creates an "acquire-waypoint" type of trajectory goal, and when the vehicle is working around obstacles or between closely-spaced waypoints, it uses a "maneuver" goal.

During execution of the acquire-waypoint goal, the command spooler directs the control system to head the vehicle toward its destination and travel forward at the rate specified in the waypoint command. The spooler pays some attention to the transitional dynamics of the vehicle as it begins to execute the goal; while the vehicle is turned away from the waypoint, the spooler commands only a very low speed, but as the vehicle's heading comes around to point toward its next waypoint the speed is increased to the commanded speed. This behavior makes the task of the bottom level activity planner simpler by ensuring that the vehicle does not travel away from its goal before it has acquired the heading to the goal.

The command spooler executes a maneuver goal by parceling out control commands to the control system. These control commands are designed to follow the trajectory specified in the maneuver goal. Each maneuver goal specifies an arc of a circle for the vehicle to follow in a horizontal plane, and a beginning and ending depth for the arc. The spooler monitors the vehicle's progress along this arc by dead-reckoning navigation, commanding a speed, heading rate, and descent rate appropriate for following

the arc until it reaches the end of the arc. Since the sensor suite of the vehicle is limited, the vehicle's speed cannot be measured directly and must be commanded open-loop. Heading and depth, however, are monitored to determine that the vehicle is following the trajectory properly, assuming that the speed is as commanded.

7. EVALUATION TESTS

The planning system has been put through a number of tests in the simulation environment to verify that it is working properly and to adjust the planning parameters to achieve good, consistent performance. Once the models and planning coordination parameters were fine-tuned, the system proved capable of good performance under the different test conditions.

Tuning the system involved determining the values to use in the replanning urgency formulae, the parameters to use in the goal planner during replanning, the threshold values by which Goal_Select determines the class of planning problem, and the values of the vehicle and environment models to use across the levels of the hierarchy to ensure their self-consistency. Compared to the task of determining the architecture's parameters, the determination of the vehicle model parameters was easy. The tests and their results are presented below.

7.1. NOMINAL MISSION

The nominal mission has been designed to test the planner's ability to perform its functions correctly under favorable conditions. By favorable conditions it is meant that no unexpected problems develop during the execution of the plan. Thus, the system should plan occasionally at the lower levels to update and maintain the plan structures, and the plan which is selected at the beginning of the mission should be executed without significant alterations.

Figure 15 shows the initial plan created at the top level before execution has begun. The geographic locations of the waypoint goals in the horizontal plane have been mapped into corresponding points (circles) on the figure. The lines connect the

waypoints in the order in which they are to be executed, starting at the bottom left corner of the figure and ending in the bottom right corner. The end goal of the plan is 100 meters to the east of the beginning goal, and the goals farthest north are 70 meters north of the beginning and ending goals. The goal planner was allocated 40 seconds to create this initial plan starting from an empty plan. Another 40 seconds were allocated between the two bottom levels in creating their initial plans, which in this case consisted of the first few goals of the top level plan.

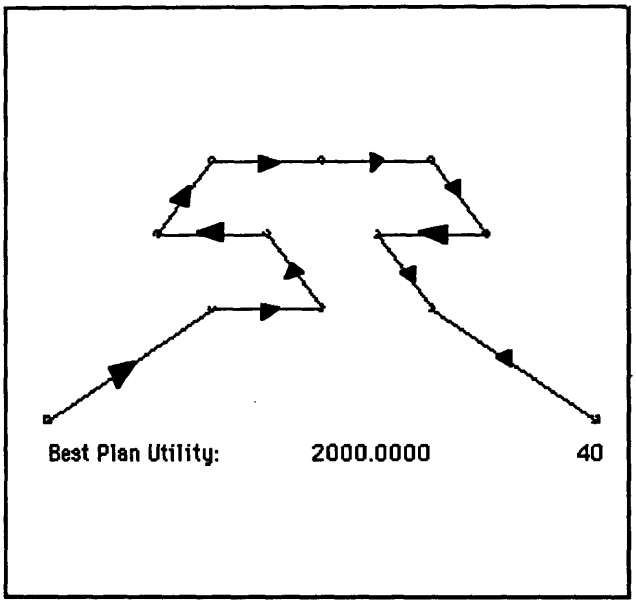

Figure 15: Initial Top-Level Plan

The progress of execution of the mission is charted in Figure 16. The figure shows the expected utility of the plans that have been created at the three levels of the hierarchy along with the actual utility (i.e., the total accumulated value) of the portion of

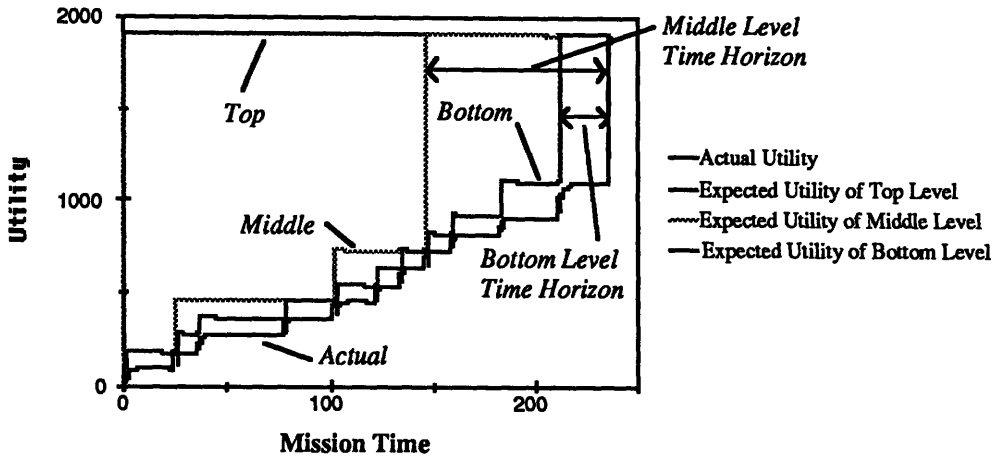

Figure 16: Expected Plan Utility at three different time horizons (top three traces) and utility of plan already complete (bottom trace)

the bottom-level plans that have been completed as a function of time. Since no portion of the plan can be executed until it has been created at the bottom level, the expected utility of the bottom level is always greater than or equal to the actual utility of the executed portion of the plan. Likewise, since the middle level plan must contain goals before they can be commanded to the bottom level plan, the middle level's expected utility is always greater than or equal to the bottom level's expected utility, and so forth. The only instance in which the bottom level's expected utility may exceed the middle level's expected utility is when the middle level has replanned but the bottom level has not yet incorporated the new plan in its estimate of utility, and similarly for the other levels.

The time horizon of a plan can be determined from the utility profiles in Figure 16. The time horizon is the horizontal distance between a point on the executed utility trace and the point at the same level of utility on the expected utility trace of a plan. The time horizon of each level varies throughout the mission due to the discreteness of plan generation and the differences in the rate at which the plan is executed and the rate at which it is expected to be executed. That the system properly maintains and updates its plans can be verified from the figure, since the plans are updated at intervals representing

consistent time horizons.

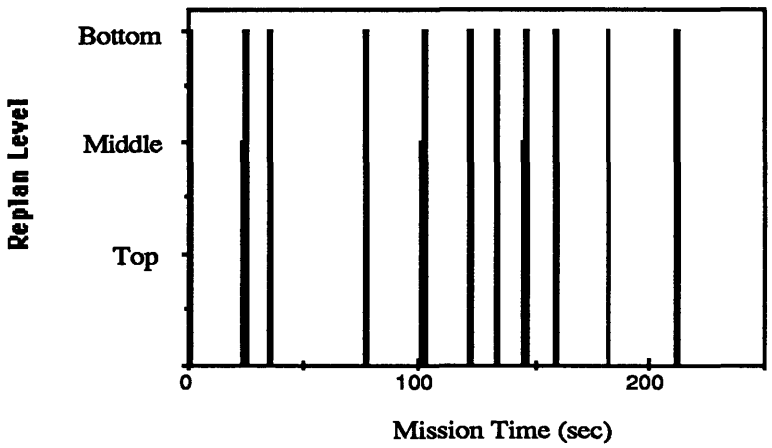

Figure 17: Goal Planner Activity For the Nominal Mission.

Figure 17 shows the periods of the mission during which the planners were actively replanning. The height of each bar indicates which level is replanning. The width of the bars indicate the length of time spent replanning at the level; however, in this mission none of the replanning took long enough to be observed with the long time scale of the horizontal axis. Figure 17 does show that only the bottom level and occasionally the middle level replanned during the nominal mission. They replanned in order to maintain the plans out to the time horizons appropriate for those levels. The top level plan was executed as originally planned.

Figure 18 shows the actual path followed by the vehicle during the course of the

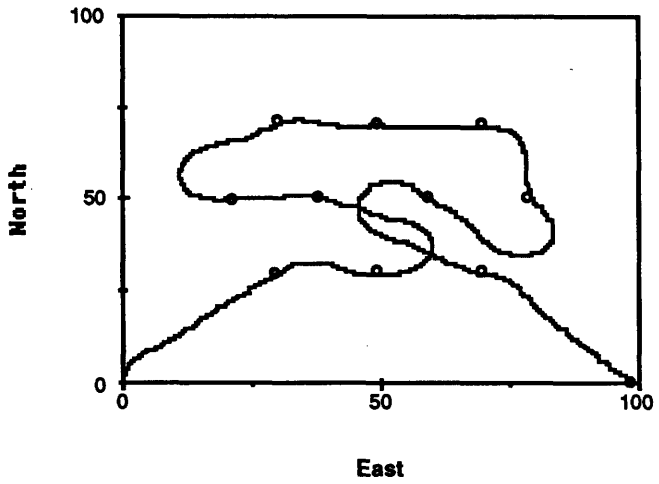

Figure 18: Actual Path of the Vehicle for the Nominal Mission

mission. Each waypoint was visited in the order specified by the original top level plan. Note that the bottom levels supply the detail necessary to command a trajectory that both passes through the waypoints that were commanded at the top level and can be followed by the control system. Note the inability of the vehicle to execute instantaneous changes in heading results in overshoot of some waypoints.

7.2. CONSTRAINED MISSION WITH UNCERTAINTY

In the constrained mission with uncertainty, the planning problem is highly constrained by limits on the vehicle's resources. In addition, there is uncertainty in the onboard models; the actual resource use is poorly modeled by the onboard resource estimation functions. Under these circumstances, small deviations from the original plan are likely to cause problems in meeting the resource constraints as planned open-loop. The planning system must recognize any forthcoming shortages or predict any unexpected resource surpluses. Replanning is initiated to ensure that the resource constraints are met through the end of the mission in the former case and that opportunities to achieve extra goals are pursued in the latter case.

The initial plan created for this mission is shown in Figure19. Figure 20 shows the replanning which occurred during the highly constrained mission. No replanning other than plan updates at the bottom levels occurred for the first 75 seconds of the mission. At 75 seconds, the top level replanned in response to an opportunity to take advantage of a predicted resource surplus. The surplus resources became available as the mission progressed because the amounts of fuel used and time required for the complete mission became more certain which allowed a reduction in the size of the resource reserves. The planner added the goal in the northwest corner of the graph, causing the vehicle to turn around and head for that goal instead of going to the end goal as it had originally planned (see Figure 21).

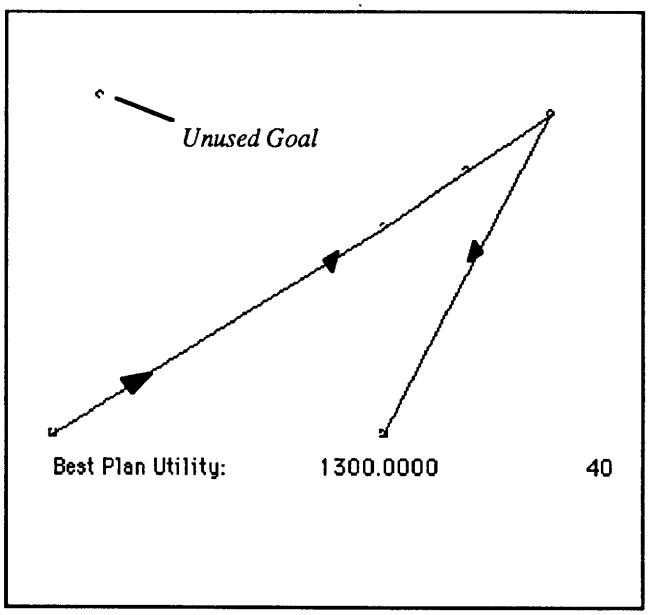

Figure 19: Initial Plan of the Highly Constrained Mission.

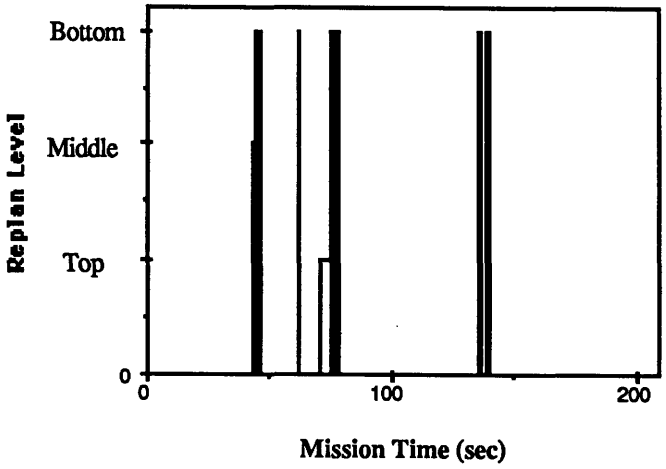

Figure 20: Replanning history of the highly constrained mission.

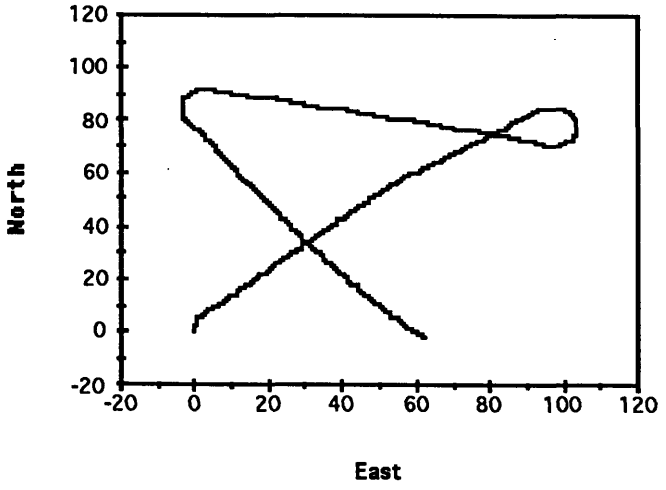

Figure 21: Vehicle Path in the Highly Constrained Mission.

The time history of the utility of the plans that is shown in Figure 22 illustrates how the uncertainties inherent in the mission affect the utility. In this mission, the resource use estimates used by the planners were somewhat pessimistic, causing the expected value of the mission to improve as execution made its resource use more certain.

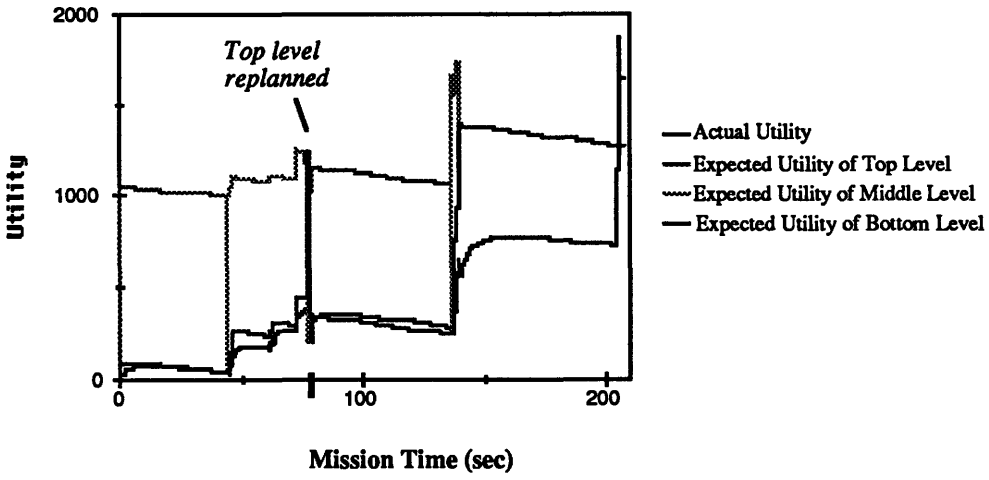

Figure 22: Utility Histories of the Highly Constrained Mission

At 75 seconds into the mission when the top level replanned, the middle level's goals no longer match the goals of the top level. Thus, estimates of the utility and resource use of the middle level plan can no longer be used to update the estimates of utility and resource use at the top level. The decrease in utility of the top level plan seen at 75 seconds in Figure 22 reflects the pessimistic top level estimates. At 140 seconds, the middle level replans and its higher estimates bolster the estimates of utility at the top level, until shortly thereafter when the bottom level replans, lowering the estimates to a more reasonable level.

The spike in the utility plot at the end of the mission shown in Figure 22 is due to achievement of the final goal of the mission. Before the final goal is achieved, its achievement is not certain because of the uncertainty in the measure of fuel left and the fact that the fuel level is near empty. When the goal is achieved, its value is added to the value of the mission. The utility traces become meaningless after the final goal is accomplished, because the system no longer updates the estimates after it has finished the mission.

7.3. MISSION WITH A DETECTED FAULT

In this mission, detection of a fault in one of the vehicle's systems is simulated. The planner's models of the vehicle's ability to perform mission goals are changed partway through the mission in the same way that a fault detection and performance monitoring system would change the models. The vehicle recognizes the effects of the fault on the outcome of the mission and replans the rest of the mission to maximize the expected value of the mission given the change in the vehicle's capabilities. Thus, the planning system devises a plan which makes the best of the situation.

In this experiment, the top-level planner created the initial plan shown in Figure 23. When the vehicle passed the first waypoint, 30 seconds into the mission, a parameter in the fuel consumption model (intended to be input from a fault-detection and performance-modeling system) was adjusted to 160% of its value, as was the corresponding factor in the simulation model which determines the actual fuel used. The fuel used in the simulation model per unit time and the fuel used in the vehicle's model per unit time are proportional to the factors which were adjusted. The immediate drop in the top level's expected utility, shown in Figure 24, indicates that the system recognized the model change.

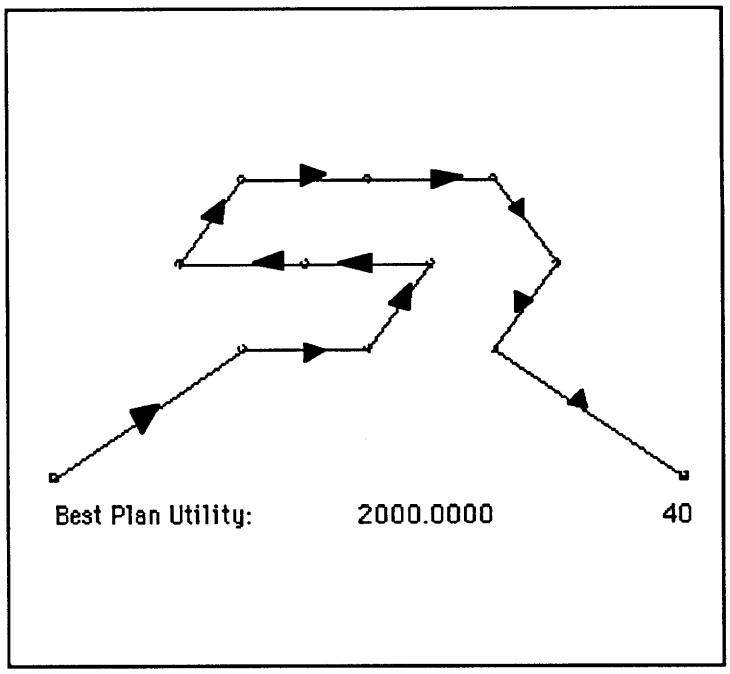

Figure 23: Initial Plan

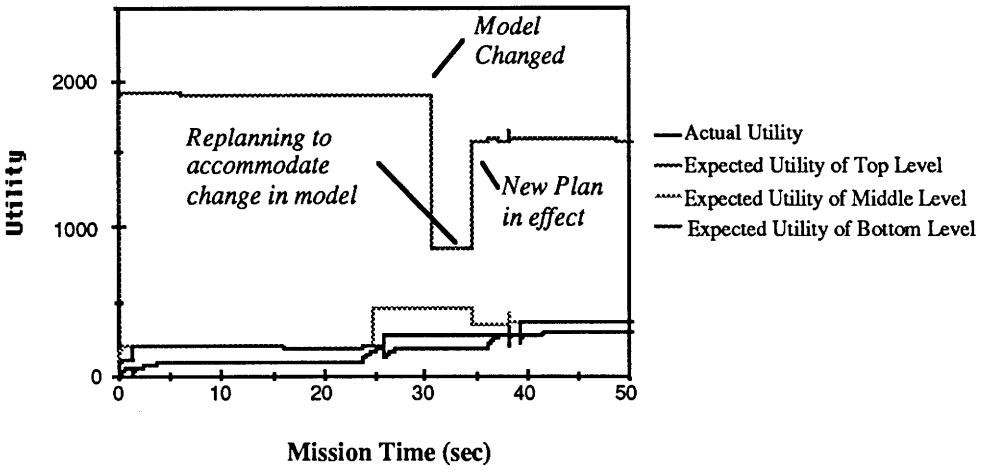

Figure 24: Utility histories of beginning of mission, 30 seconds into which a fault was detected.

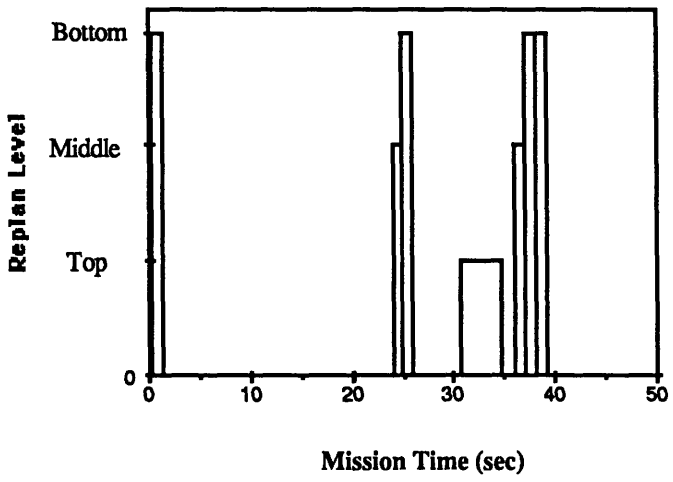

Figure 25: Planning history of the beginning of the mission in which a fault was detected.

Figure 25 shows that the top level replanned the mission immediately, and upon completing its replanning the expected utility of the top level in Figure 24 increased nearly to its original level. Figure 26 shows the top-level plan which replaced the initial plan after the model change; the planner dropped several goals from the plan resulting in a feasible plan given the revised fuel consumption model. The new plan was spliced onto the old plan at the point in the old plan which had just been completed, resulting in the vehicle path over the course of the mission shown in Figure 27. The second level did not need to replan immediately since the next goal of the old top level plan and the first goal of the new top level plan match, meaning that the first part of the current middle level's plan was still valid.

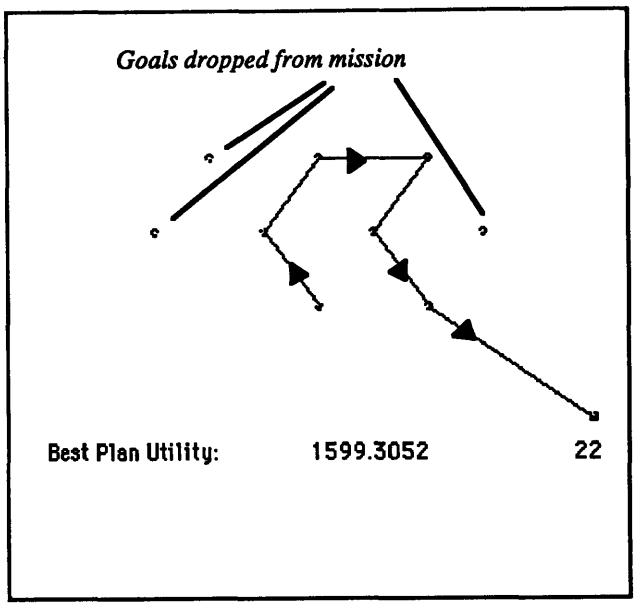

Figure 26: Three goals were dropped from the plan to allow successful completion of the mission.

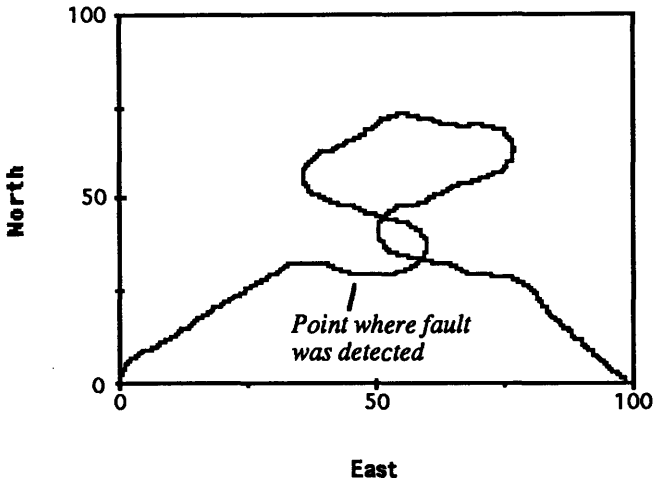

Figure 27: Path followed by the vehicle over the course of the mission.

Figures 28 and 29 show the entire detected-fault mission. The adjusted models used by the planner to estimate resource use underestimate the resources which will be used by about 4%. At the end of the mission, the vehicle has very little of its constrained resources left and is using them at a faster rate than expected. Not only does execution not follow the plan precisely in terms of its resource use, but the resource use approaches

the constraints; these factors couple to produce a decreasing expectation of utility as the mission finishes. The planning system responds to the rapidly-changing expectations by replanning repeatedly.

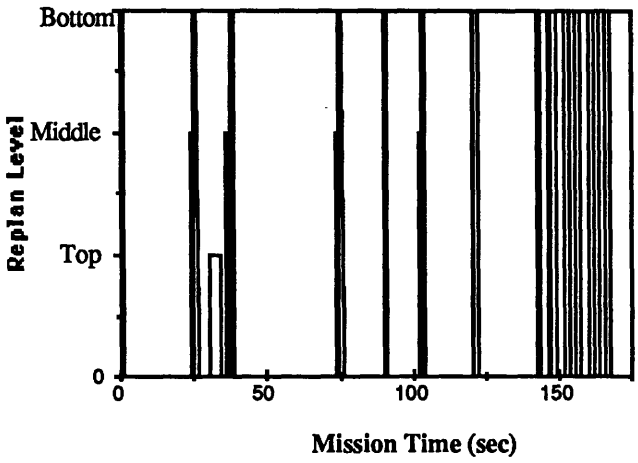

Figure 28: Planning history of the entire mission.

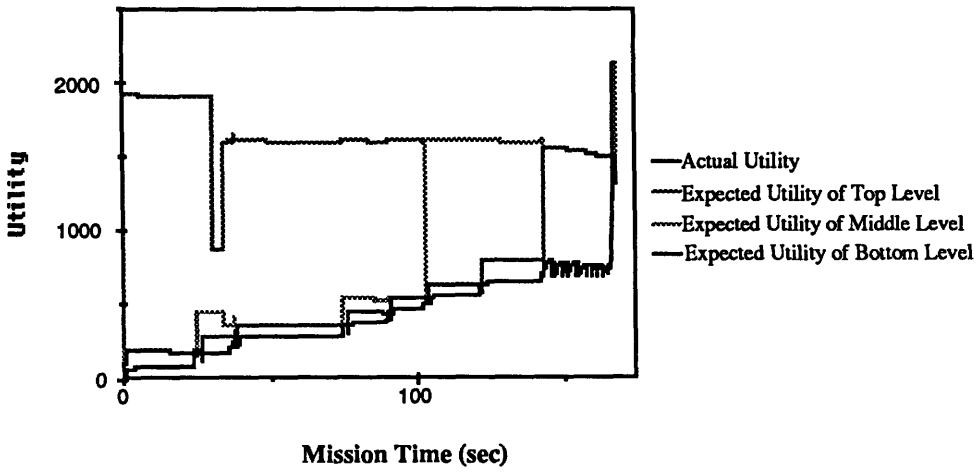

Figure 29: Utility history of the mission.

7.4. MISSION WITH AN UNDETECTED FAULT

This experiment shows the performance of the planning system when the models used by the planner are not well-matched to the vehicle's actual performance. Although an undetected fault, such as a fouled thruster or a change in hydrodynamic characteristics due to a damaged vehicle, affects the vehicle's performance drastically, the vehicle's models which are internal to the planning system are not updated to reflect the changed performance in this example. The best the planner can be expected to do in this circumstance is to replan the mission to account for the observed poor performance of that segment of the mission which has already been executed. This experiment is conducted to highlight the performance of the planning system under conditions in which the models used by the planner are poor and the fault detection and performance monitoring functions do not correct them. In fact, the fault detection and monitoring functions would have to be poorly designed not to have corrected at least partially for the substantial change in performance observed in this experiment.

In this mission, the time and fuel use of the vehicle are doubled from the expected values but the vehicle's onboard performance models remain unchanged. The initial top-level plan is shown in Figure 30, where the constraints on the vehicle's resources have prevented the planner from including all of the potential waypoint goals in its plan. As the vehicle approaches the first goal, the unexpected high rate of resource use

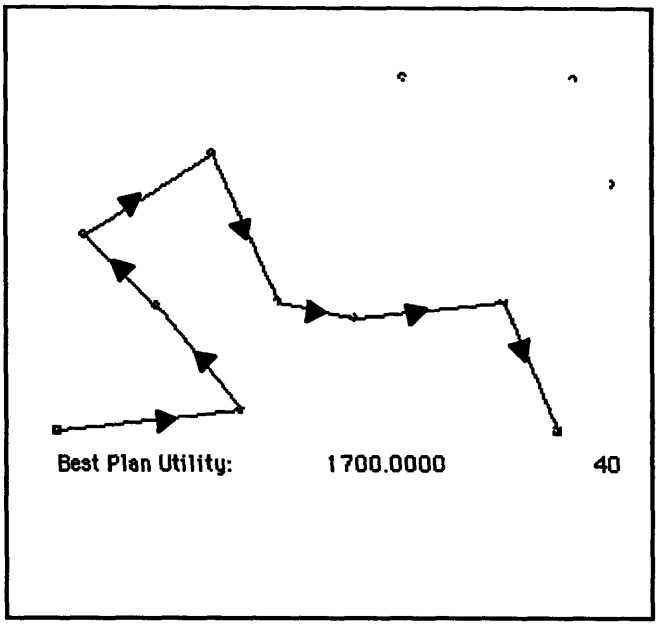

Figure 30: Initial plan of the mission with an undetected fault.

has made the difference between the current state of the vehicle and the expected state of the vehicle large enough that the system's bottom level is forced to replan. That is, the allocated resources of the bottom level have been exceeded or nearly exceeded by the actual resource use. Figure 31 shows the replanning activity at the bottom level approximately 25 seconds into the mission. Had the replanning been caused by the need to generate more plan, planning would not have taken so long and the associated replanning trace at 25 seconds would have been narrower.

The bottom level is the appropriate level for replanning at this point since it is at the bottom level that the expectation of resource use has changed the most, as a fraction of the allocated resources. Had the vehicle models properly accounted for the increase in resource use continuing into the future, the top level would have replanned instead as it did in the detected-fault experiment.

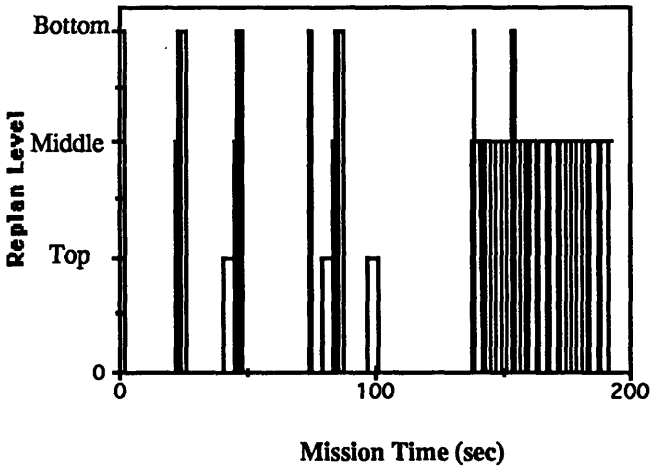

Figure 31: Planning History For the Undetected-Fault Mission.

Shortly after 40 seconds into the plan, the top level assesses the reduced value of the overall plan due to the excess resource use. The top level has replanned before the middle level in this case because the expected utility of the top level, as a fraction of the utility originally expected, has decreased more quickly than that of the middle level. This large drop in utility is a combined result of the high importance placed on the final goal by the top level plan and the falling probability of completing the final goal. At 40 seconds on Figure 32, the top level utility is observed to have dropped significantly, leading to the replan. The new top level plan is shown in Figure 33.

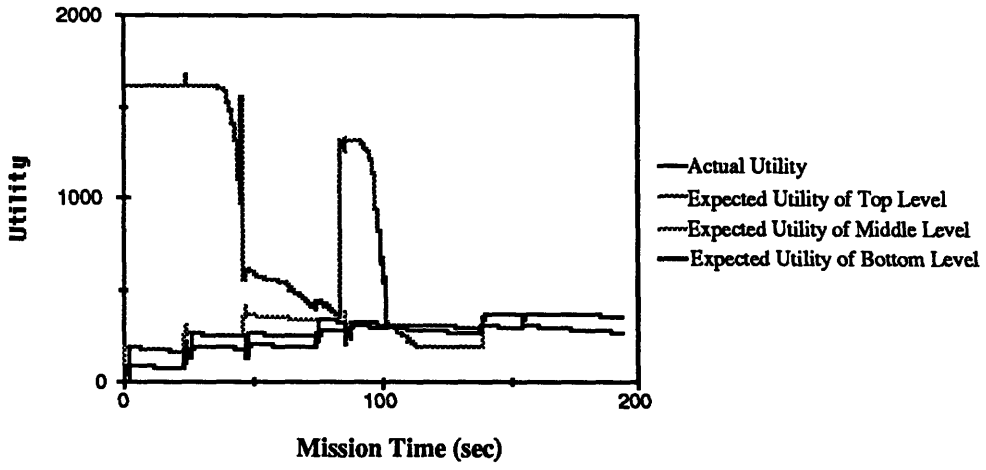

Figure 32: Utility History For the Undetected-Fault Mission.

Immediately following the top level's replan in Figure 31, the second level replans to incorporate the new top level plan. Next, the bottom level must replan. The spike in the top level's expected utility immediately after the replan is an artifact of the way the plan estimates are updated. The resource and utility estimates from the lower levels are not incorporated into the top level utility until the lower levels have incorporated the new top level plan. Therefore, when the bottom level finishes incorporating the new goals into its plan, the expected utility of the top level drops to its correct value.

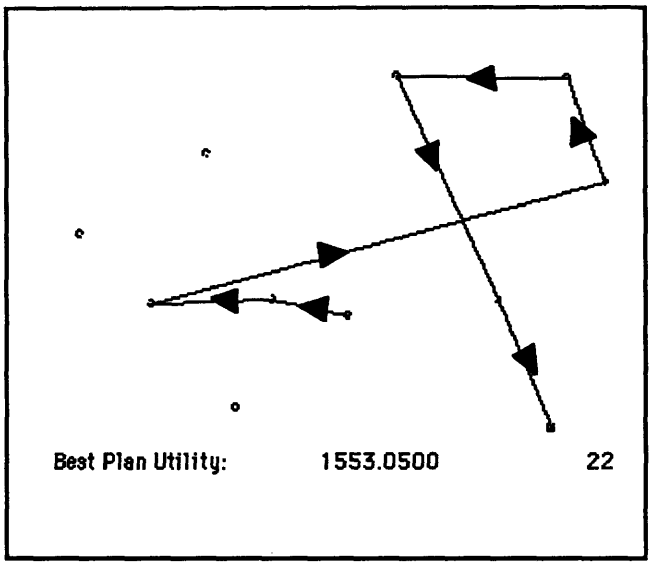

Figure 33: Second Plan of the Undetected-Fault Mission

Execution of the plan continues to reduce the expected utilities across all planning levels, causing the top level to replan again at 75 seconds and one last time at 100 seconds. The results of the last two top level replans are shown in Figures 34 and 35. In the fourth replan, the top level's models indicate that it will no longer be able to reach the final goal, so it plans to accomplish as much as it possibly can. After the fourth replan, no more replans are performed since the top level's plan can not degrade much further than it already has. Since the lower level plans are normalized against smaller expected utility values, they are more sensitive to small changes in plan utility than the upper level. Thus, during the last stages of the mission the low levels replan because of the higher-than-expected resource use.

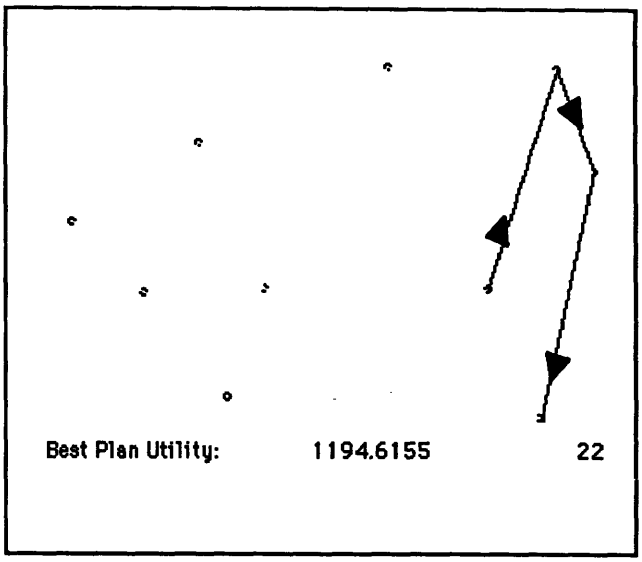

Figure 34: Third Plan of the Undetected-Fault Mission

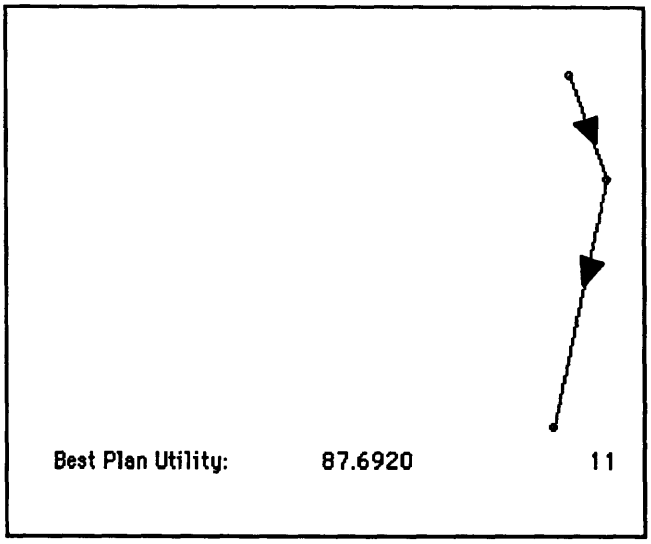

Figure 35: Final Plan of the Undetected-Fault Mission.

Figure 36 shows the result of all the replanning. The vehicle began following an ambitious plan which it had to pare down several times during the course of the mission as it realized that it was not able to accomplish the entire plan. The path of the vehicle shows that the vehicle changed its plan in mid-execution, causing the loop in the path at (30,60) (north,east) and a rough transition between the next-to-last and the last plans at

(30,90).

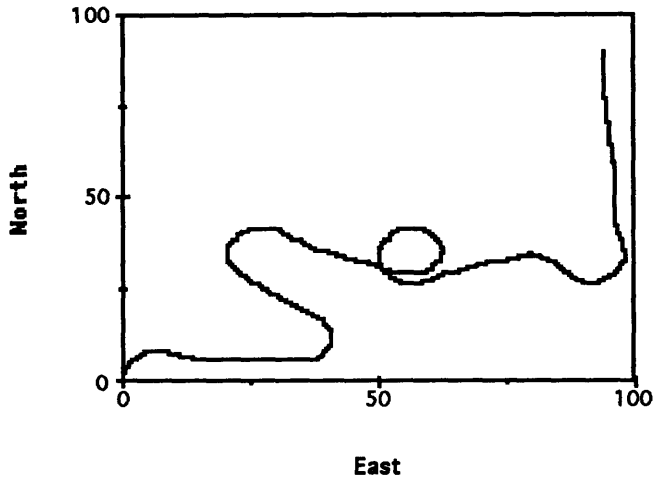

Figure 36: Vehicle path over the course of the undetected-fault mission.

8. CONCLUSION

The proposed hierarchical architecture was designed, implemented, and successfully tested in the NetSim simulation environment for an autonomous underwater vehicle problem. The architecture was shown to provide a framework for closed-loop autonomous planning and planning management. As execution progresses, plans are updated at the three levels of a hierarchical planning system; the execution of the plans is monitored, and replanning is initiated when plans can not proceed as intended.

Some of the results of the tests in Chapter 7 may not be perfect. However, the intent of this thesis is to justify the structure of the hierarchical planning architecture presented herein, not the particular algorithms, functions, and constants employed in this implementation of the architecture. The particular algorithms, functions, and constants used here are intended to fulfill the minimal requirements of the architecture; they need considerable improvement to realize the full potential of the architecture. The structure of the architecture should not be dismissed because of shortcomings in the results that are due to a lack of better algorithms.

8.1. MERITS OF APPROACH

The architecture developed here is based on aspects of an autonomous planning problem which are not specific to any particular type of autonomous mobile robotic system. Indeed, it is based on features of planning problems common to many systems. The system has the restriction that the value of its objectives must be additively separable given the state of the system and the state of the environment at the beginning of the execution of each objective.

When applying the architecture to solve the planning problem for a specific

autonomous system, the problem must be represented within the architecture in terms of models of resource use and a hierarchical decomposition of each of the system's goals. The resource use and conditions for completion of each goal are defined explicitly in the resource models. The models explicitly describe the system in terms of the specific decomposition of system goals into system actions and in terms of the use of the system's constrained resources caused by executing the mission goals. This explicit description of the system makes its evaluation straightforward and its performance as predictable as possible, since the relationships among the system goals and their expectations of the system are clearly defined. The small size of the set of goals and models used to implement the autonomous vehicle transit goals demonstrated here illustrate the effectiveness of the strategy.

8.2. SUGGESTION FOR FURTHER DEVELOPMENT

Comparison of the results of test cases three and four show that the potential of the architecture is only achieved if the models of the specific autonomous system are accurate. Since the planning architecture uses the models of the autonomous system to evaluate its performance during execution, incorrect or poorly adjusted models cause the planning system to make sub-optimal plans and to have difficulty with their execution. The use of a fault-detection and performance monitoring system is required to enhance the performance of the planner by correcting the onboard performance and resource use models in response to actual changes in performance.

The strength and generality of the architecture could be further improved if some form of learning could be applied to the models of vehicle performance to improve them as the mission progressed. The application of connectionist learning techniques (such as those employed in control problems to improve the model of the dynamics of the plant to be controlled [24]) to the models could also reduce the difficulty of providing precise

models of the system's performance when configuring the architecture to a given autonomous system by eliminating much of the necessary fine-tuning.

9. REFERENCES

[1] A. J. Healey, R. B. McGhee, R. Cristi, F. A. Papoulias, S. H. Kwak, Y. Kanayama, Y. Lee, "Mission Planning, Execution, and Data Analysis for the NPS AUV II Autonomous Underwater Vehicle", *Proceedings of Mobile Robots for Subsea Environments; International Advanced Robotics Program,* Monterey, CA, 1991.

[2] R. Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation,* Vol. 2, 1986.

[3] J. G. Bellingham, T. R. Consi, R. M. Beaton, W Hall, "Keeping Layered Control Simple", *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology,* Washington, DC, 1990.

[4] J. G. Bellingham, T. R. Consi, "State Configured Layered Control", *Proceedings of Mobile Robots for Subsea Environments; International Advanced Robotics Program,* Monterey, CA, 1991.

[5] J. K. Rosenblatt, D. W. Payton, "A Fine-Grained Alternative to the Subsumption Architecture for Mobile Robot Control", *International Joint Conference on Neural Networks,* Washington, DC, 1989

[6] D. W. Payton, "An Architecture for Reflexive Autonomous Vehicle Control", *Proceedings of the IEEE Conference on Robotics and Automation,* San Francisco, CA, 1986

[7] O. J. Rodseth, "Software Structure for a Simple Autonomous Underwater Vehicle", *Proceedings of the Symposium on Autonomous Underwater Vehicle Technology,* Washington, DC, 1990.

[8] O. J. Rodseth, "Object Oriented Software System for AUV Control", *Proceedings of Mobile Robots for Subsea Environments; International Advanced Robotics Program,* Monterey, CA, 1991.

[9] A. Kramer, D. Toms, R. Schrag, D. Johnson, "Operational Planning and Programming of Autonomous Underwater Vehicles", *Sixth International Symposium on Unmanned Untethered Submersible Technology,* Durham, NH, 1989

[10] G. T. Russell, R. M. Dunbar, "Intelligent Control and Communication Systems for Autonomous Underwater Vehicles", *Proceedings of Mobile Robots for Subsea Environments; International Advanced Robotics Program,* Monterey, CA, 1991.

[11] J. L. Allison, D. P. Watson, T. M. Cook, "Intelligent Waypoint Transiting in Complex AUV Environs", *Sixth International Symposium on Unmanned Untethered Submersible* Technology, Durham, NH, 1989

[12] R. Alami, R. Chatila, P. Freedman, "Task Level Teleprogramming for Intervention Robots", *Proceedings of Mobile Robots for Subsea Environments; International Advanced Robotics Program,* Monterey, CA, 1991.

[13] X. Zheng, E. Jackson, M. Kao, "Object-Oriented Software Architecture for Mission-Configurable Robots", *Proceedings of Mobile Robots for Subsea Environments; International Advanced Robotics Program,* Monterey, CA, 1991.

[14] J. Bares, M. Hebert, T. Kanade, E. Krotkov, T. Mitchell, R. Simmons, W. Whittaker, "Ambler; An Autonomous Rover for Planetary Exploration", *Computer,* June 1989, pp 18-26.

[15] J. Hultman, A. Nyberg, M. Svensson, "A Software Architecture for Autonomous Systems", *Sixth International Symposium on Unmanned Untethered Submersible Technology,* Durham, NH, 1989

[16] J. S. Albus, *System Description and Design Architecture for Multiple Autonomous Undersea Vehicles,* NIST Technical Note 1251, Washington, DC, Sept. 1988.

[17] Mesarovic, Mihajlo D., D. Macko, and Y. Takahara, *Theory of hierarchical, multilevel, systems,* Academic Press, New York, 1970.

[18] Adams, M.B., Harrison, J.V., Deutsch, O.L., "A Hierarchical Planner for Intelligent Systems," *Proceedings of the SPIE Conference on Applications of Artificial Intelligence,* Crystal City, VA, April 1985.

[19] Singh, M.H. and Titli, A., *Systems: Decomposition, Optimization and Control,* Pergamon Press, Oxford, 1978.

[20] Schweppe, Fred C., *Uncertain Dynamic. Systems,* Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1973.

[21] Sikora, Scott, *Implementation Issues in Hierarchical Planning and Scheduling Problems,,* CSDL-T-1100,

Charles Stark Draper Laboratory, Cambridge, MA, 1991.

[22] Haimes, Y. Y., Tarvainen, K., Shima, T., Thadathil, J., *Hierarchical Multiobjective Analysis of Large-Scale Systems*, Hemisphere Publishing Corporation,, New York, 1990.

[23] Yoerger, D. R., Bradley, A. M., Walden, B. B., "The Autonomous Benthic Explorer (ABE): An AUV Optimized for Deep Seafloor Studies", *7th International Symposium on Unmanned Untethered Submersible Technology*, Univ. of New Hampshire, June 1991.

[24] Nistler, Noel F., *A Learning Enhanced Flight Control System for High Performance Aircraft*, S.M. Thesis, CSDL-T-1127, Charles Stark Draper Laboratory, Cambridge, MA, 1992.