In Neural Networks for Control, Miller, W.T. and Sutton, R., eds., MIT Press: Cambridge, MA, 1990

Intelligent Control for Multiple Autonomous Undersea Vehicles

Martin Herman, James S. Albus and Tsai-Hong Hong

Robot Systems Division
National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, MD 20899

ABSTRACT

Intelligent control for autonomous vehicles in a natural, potentially hostile environment requires a system that integrates artificial intelligence with modern control theory, and that is implemented on parallel, possibly special-purpose hardware. Issues dealing with the requirements of such a system are discussed in the context of the Multiple Autonomous Undersea Vehicles (MAUV) project. The MAUV control system and its implementation are also presented.

The goal of the MAUV project was to have multiple undersea vehicles exhibiting intelligent, autonomous, cooperative behavior. The MAUV control system is hierarchically structured and incorporates sensing, world modeling, planning and execution. The levels in the hierarchy include: mission, group, vehicle task, elemental action, primitive action, and servo. Issues of real-time planning and dynamic replanning in unstructured environments are discussed. A multi-level world model that supports real-time planning is also described. Finally, timing issues, implementation, and initial experimental results are presented.

Intelligent Control for Multiple Autonomous Undersea Vehicles

Martin Herman, James S. Albus and Tsai-Hong Hong

Robot Systems Division
National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, MD 20899

1. Introduction

In order to achieve real-time intelligent control of multiple autonomous vehicles in complex environments, research issues such as distributed control, knowledge-based systems, real-time planning, world modeling, value-driven reasoning, intelligent sensing, intelligent communication, gaming, cooperative problem solving, and learning must be addressed. The types of activities that must be achieved by these autonomous systems include aggression, predation, exploration, stealth, deception, escape, communication and cooperation. These activities are required in order to thrive in a natural and potentially hostile environment.

The goal of the NIST Multiple Autonomous Undersea Vehicles (MAUV) project was to examine some of these issues in the underwater domain by attempting to achieve intelligent, autonomous, cooperative behavior in multiple vehicles. Our approach was to develop a control system architecture that fully integrates concepts of artificial intelligence with those of modern control theory. A first cut at algorithms and software to implement this architecture was developed, and this software was downloaded into computer boards mounted on board the vehicles. A series of demonstration tests was then planned for two undersea vehicles in Lake Winnipesaukee in New Hampshire.

After presenting the MAUV vehicles and scenarios, this paper provides a discussion of issues dealing with intelligent control and then presents a hierarchical control system architecture which addresses these issues. The application of this control architecture to the MAUV vehicles is then described, along with how planning, execution and world modeling are accomplished. Finally, timing issues, implementation, and experimental results are described. Further details on the project may be found in [2].

1.1. The MAUV Vehicles

Figure 1a shows a diagram, and Figure 1b a photograph, of a MAUV vehicle. These vehicles were designed and constructed by the Marine Systems Engineering Laboratory at the University of New Hampshire. They are a derivative of the EAVE-EAST vehicle [3] developed at the same lab. The vehicle is gravity stabilized in pitch and roll, with thrusters that allow it to

The MAUV project was funded by the Defense Advanced Research Projects Agency (DARPA) Naval Technology Office.

Identification of commercial equipment in this paper is only for adequate description of our work. It does not imply recommendation by the National Institute of Standards and Technology, nor that this equipment was necessarily the best available for the purpose.

EAVE III

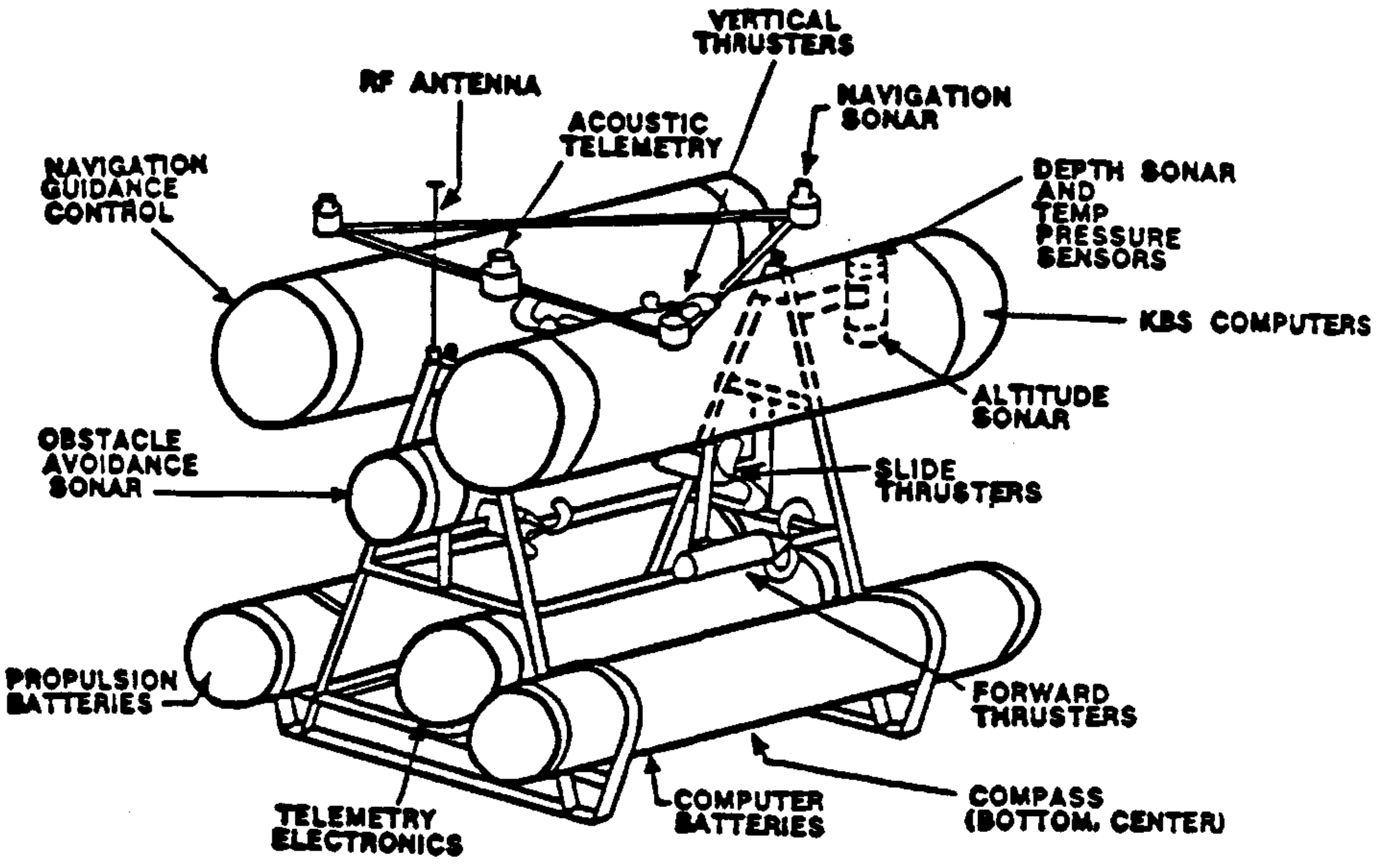


Figure 1a.

Diagram of University of New Hampshire EAVE-EAST vehicle.

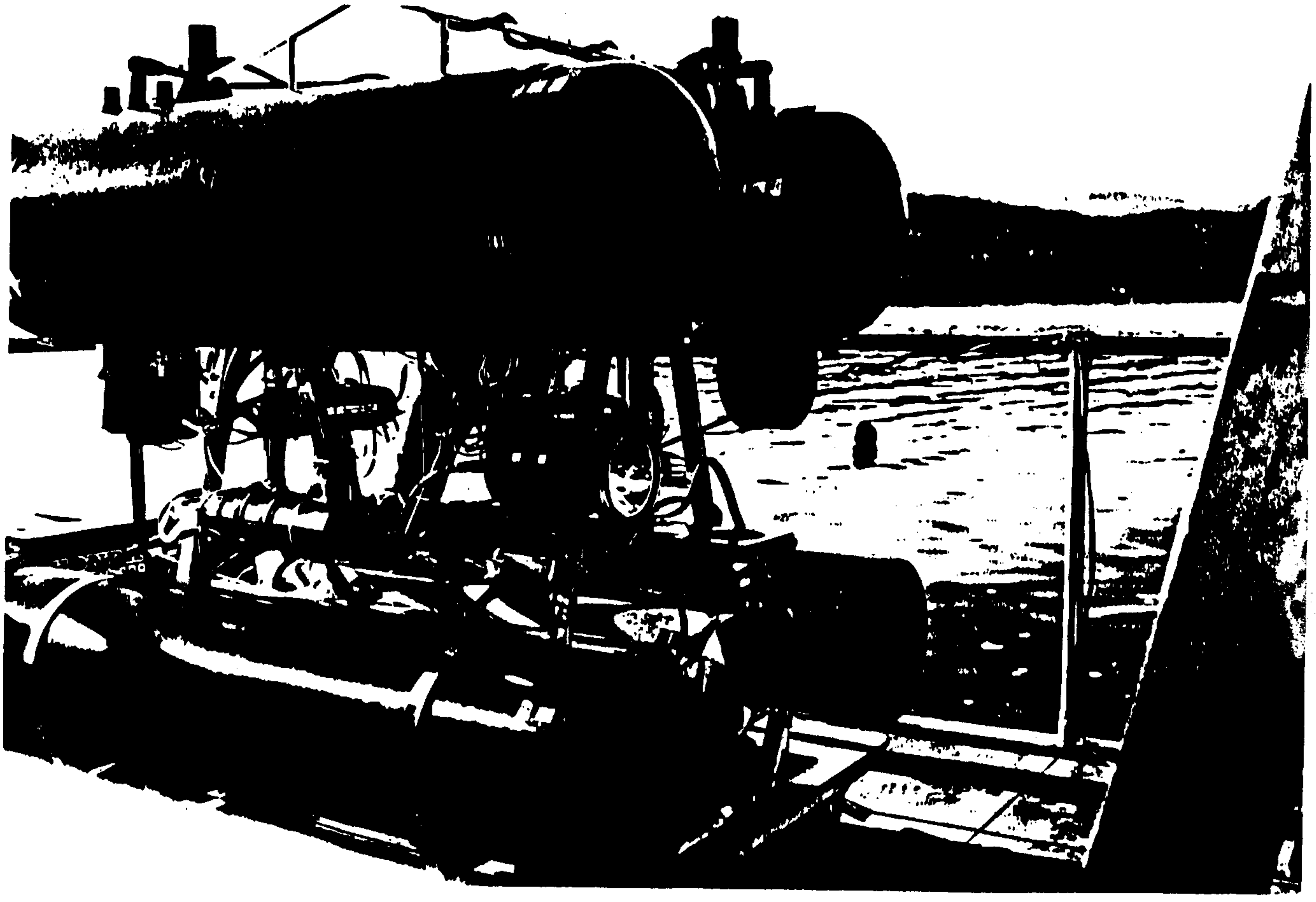


Figure 1b.

[illegible] University of New Hampshire EAVE-EAST MAUV vehicle.

be controlled in x, y, z, and yaw. It is battery powered with the batteries stored in cylindrical tanks at the bottom of the vehicle. The vehicle carries three acoustic navigation transponders which are configured as an equilateral triangle. Each transponder operates on a different frequency and different turnaround time. They receive acoustic signals from navigation bouys placed in the water, allowing range and bearing relative to these bouys to be measured. The vehicle carries a compass, pressure and temperature sensors, and depth and altitude sonars. In front,· it has an obstacle avoidance sonar consisting of five narrow beam acoustic transmitter-receivers. These are arranged such that the center sonar beam points straight ahead, two point ten degrees to the right and left, and two point ten degrees up and down from the center beam. In addition, the vehicle carries both acoustic and radio telemetry systems. All computer boards are mounted in card cages inside the flotation tanks at the upper part of the vehicle.

1.2. Scenarios

The MAUV project planned to conduct a series of demonstrations by two vehicles. These tests centered around two scenarios -- cooperative search and cooperative near-target maneuvers. The search scenario involves traversing an area either to map it out or to seek targets. Figure 2 shows a search plan that involves transiting from a base at the island to a search area and then performing a raster search of the area. The vehicles may be either near the water surface or near the lake bottom when performing the search. The concept of using two or more vehicles to search and map shallow areas is shown in Figure 3. In this scenario, the vehicles were to demonstrate the ability to measure the bottom topology, and to search for and map the positions of objects on the bottom and in the water. The vehicles were to execute a variety of search patterns, including several involving separation and rendezvous for exchange of information. The vehicles were to compute maneuvering tactics which take into account bottom toplogy and simulated enemy positions. The vehicles were to demonstrate the use of topological maps of the bottom for local navigation, and were to use both visual and acoustic bottom sensors to update these maps in real time. Obstacle avoidance sonar and bottom altitude sonar were to give the vehicles the ability to follow bottom topographic features such as ravines and ridges. The vehicles planned to demonstrate tactics using bottom features for shadowing their movement from enemy positions.

The near-target maneuvers scenario involves performing triangulation maneuvers near a target either to localize it or to take pictures of it. Figure 4 shows how target localization occurs. The two vehicles, either while patrolling or while performing a search, detect a target in direction *beta* using passive sonar. (Passive sonar involves detection of noise originating at the target.) Passive detection gives only direction but no range information. At this point, the vehicles determine two positions perpendicular to and equidistant from the line *beta*, and each vehicle travels to its position. The vehicles can then emit sonar pulses and use triangulation to accurately localize the target. In a separate scenario, the vehicles use similar maneuvers to achieve the triangle configuration, and then one vehicle illuminates the target while the other vehicle takes pictures. Having a light source some distance away from the camera, and being able to vary the position of this light source relative to the camera, can often greatly enhance undersea photography.

A third scenario being considered for the MAUV vehicles was rendezvous and docking. This might involve using sonar for the two vehicles two rendezvous with one another, and optical tracking methods for docking. Both side-by-side and end-to-end docking can be considered.

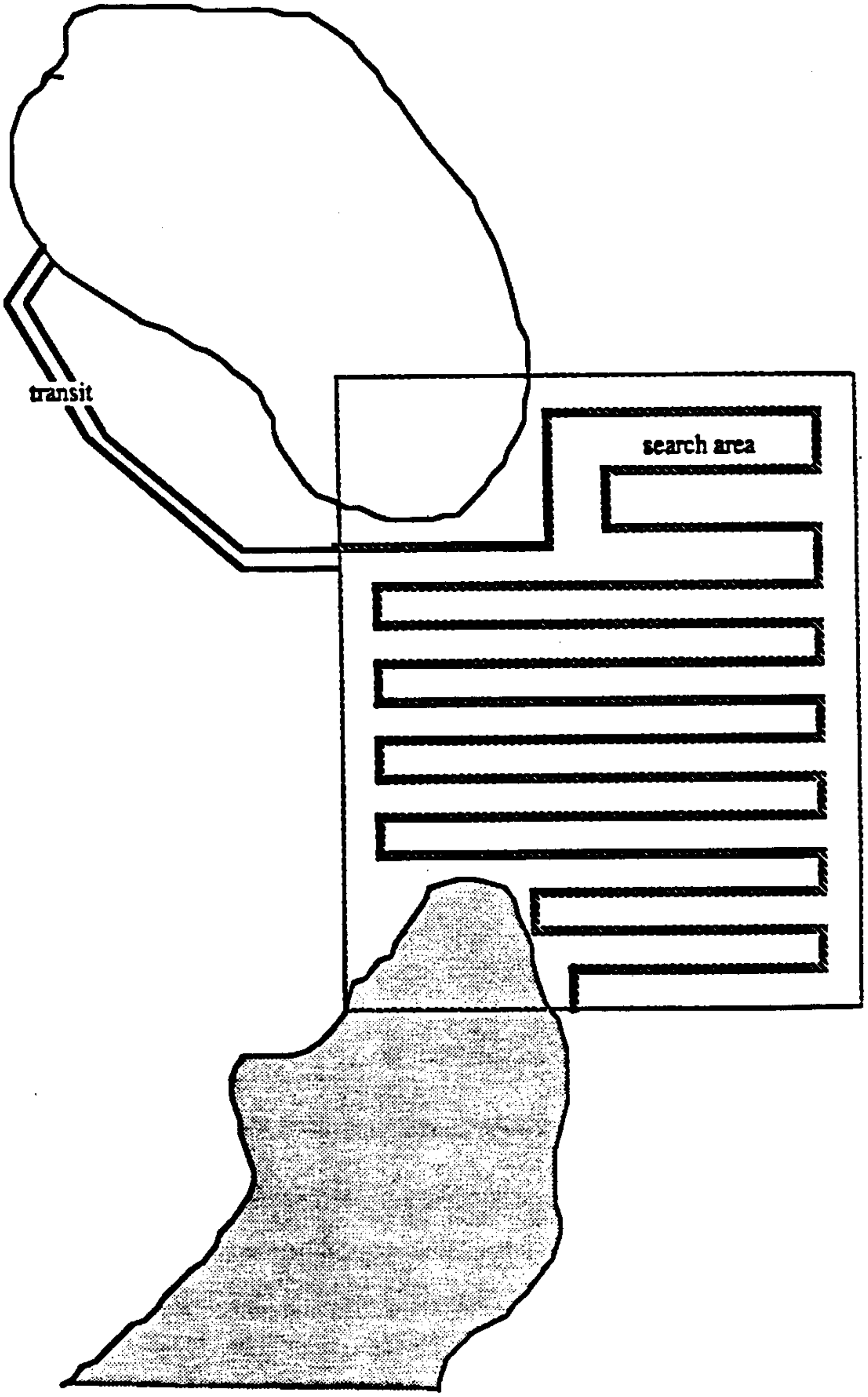


Figure 2.
Raster search

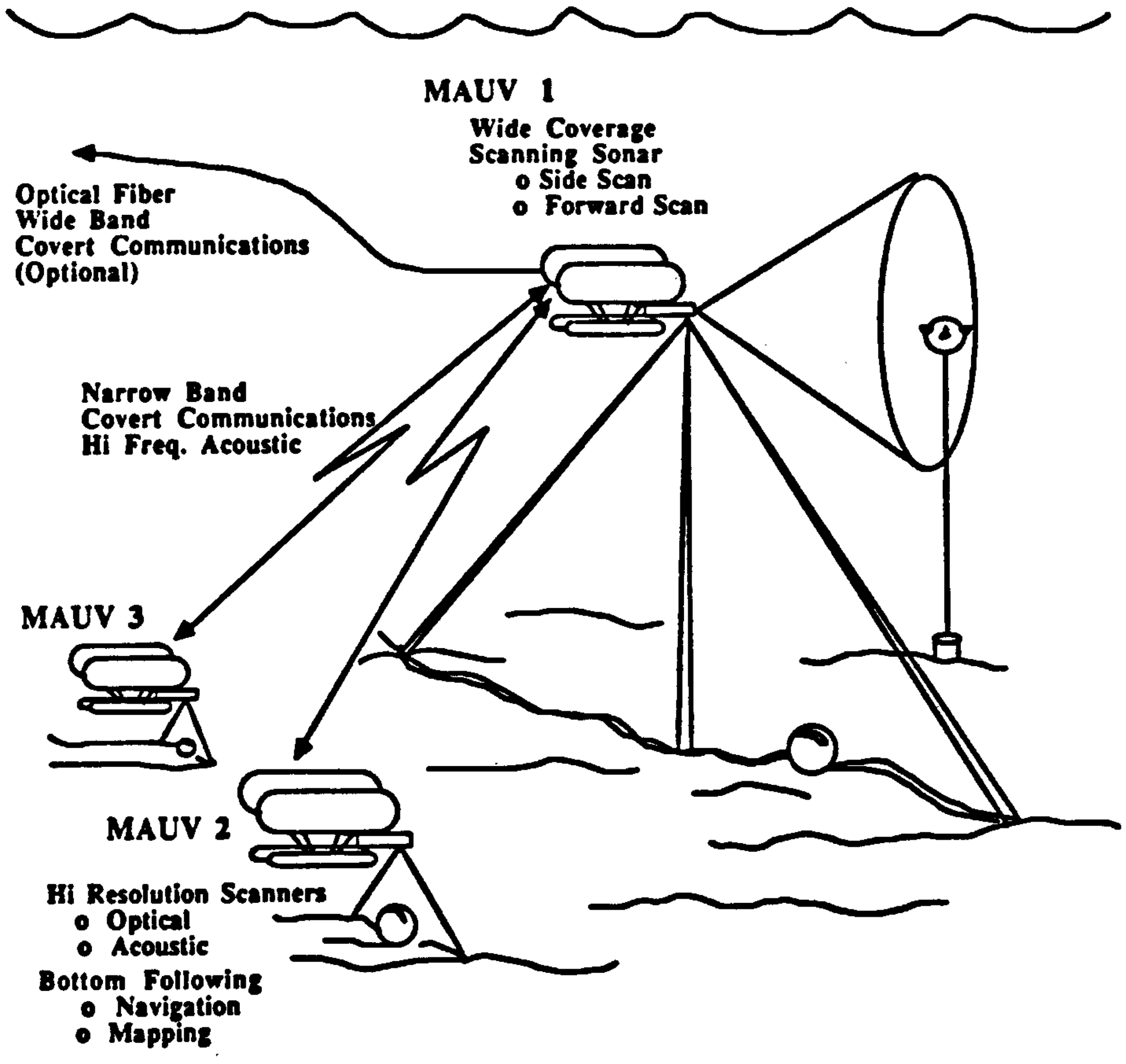


FIGURE 3. Illustration of MAUV search and map scenario.

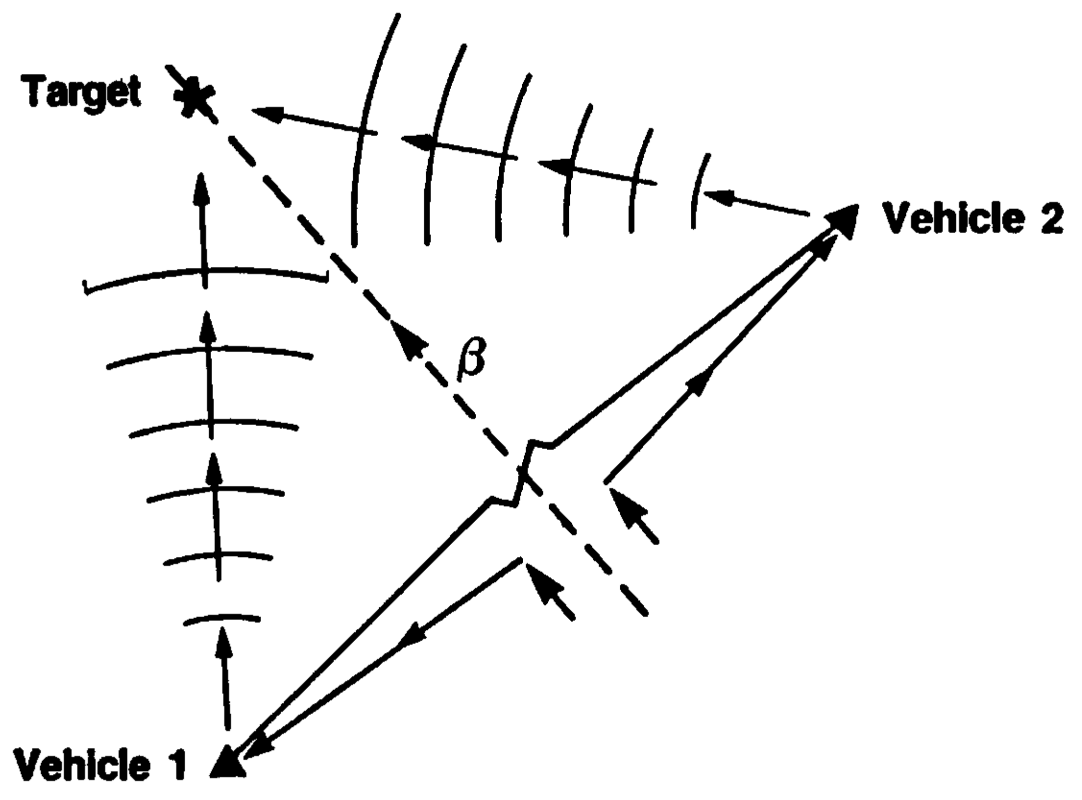


Figure 4.

Target localization using triangulation.

2. Intelligent Control for Autonomous Vehicles

Autonomous vehicles that operate in complex environments require intelligence. Truly intelligent machines will have complex system architectures in which sensing, acting, sensory processing, world modeling, task decomposition, value judgements, and goal selection are integrated into a system which responds in a timely fashion to stimulation from the environment. Figure 5 illustrates the basic elements of an intelligent control system:

1. ACTUATORS -- Within any intelligent system there are actuators which move, exert forces, and position arms, legs, hands and eyes. For an intelligent vehicle, actuators generate forces to point sensors, excite tranducers, and steer locomotion. The actuators are motors, pistons, valves, solenoids and transducers.

2. SENSORS -- Sensors for an intelligent vehicle may include vision, position, distance, vibration, acoustic, pressure, and temperature measuring devices. Sensors may be used to monitor both the state of the external world and the internal state of the vehicle itself. Sensors provide input to a sensory processing system.

3. SENSORY PROCESSING -- An intelligent sensory processing system compares observations with expectations generated by an internal world model. Sensory processing algorithms perform both temporal and spatial integration, so as to detect events and recognize features, objects, and relationships in the world. Sensory input data from a variety of sensors over extended periods of time are fused into a consistent unified perception of the state of the world. Sensory processing algorithms may compute distance, shape, orientation, surface characteristics, and material properties of objects and regions of space.

4. TASK DECOMPOSITION -- An intelligent system has processes which decompose high level goals into low level actions. Task decomposition involves both the planning and execution of actions. It requires the ability to reason about geometry and dynamics, and to formulate or select plans based on values such as cost, risk, utility, and goal properties. Task planning and execution must often be done in the presence of uncertain, incomplete, and sometimes incorrect information. The execution of tasks must be monitored and existing plans must be modified whenever the situation requires. Task decomposition is a hierarchical process requiring a multiplicity of planners that simultaneously generate and coordinate plans for many different subsystems with different planning horizons and different degrees of detail at each hierarchical level.

5. WORLD MODEL -- The world model is the intelligent system's best estimate of the state of the world. The world model includes a database in which is stored knowledge about the world. It provides information about the state of the world to the task decomposition system so that it can make intelligent plans and behavioral choices. It also provides expectations and predictions to the sensory processing system in order to enhance its ability to analyze sensory data. The world model is kept up-to-date by the sensory processing system.

6. VALUES -- Any intelligent system must have a value system in order to make judgements as to what is good and bad. The value system must evaluate both the observed state of the world and the predicted results of hypothesized plans. It must compute costs, risks, and benefits of observed situations and of planned activities. Without a means of making value judgements, an intelligent task decomposition system has no basis for choosing one action over another.

7. GOAL SELECTION -- Goals are selected by a looping interaction between the goal selection, world model, and value systems. The goal selection system hypothesizes actions, the world model predicts probable results, and the value system evaluates the predicted results. The goal selection system then chooses the hypothesized action with the

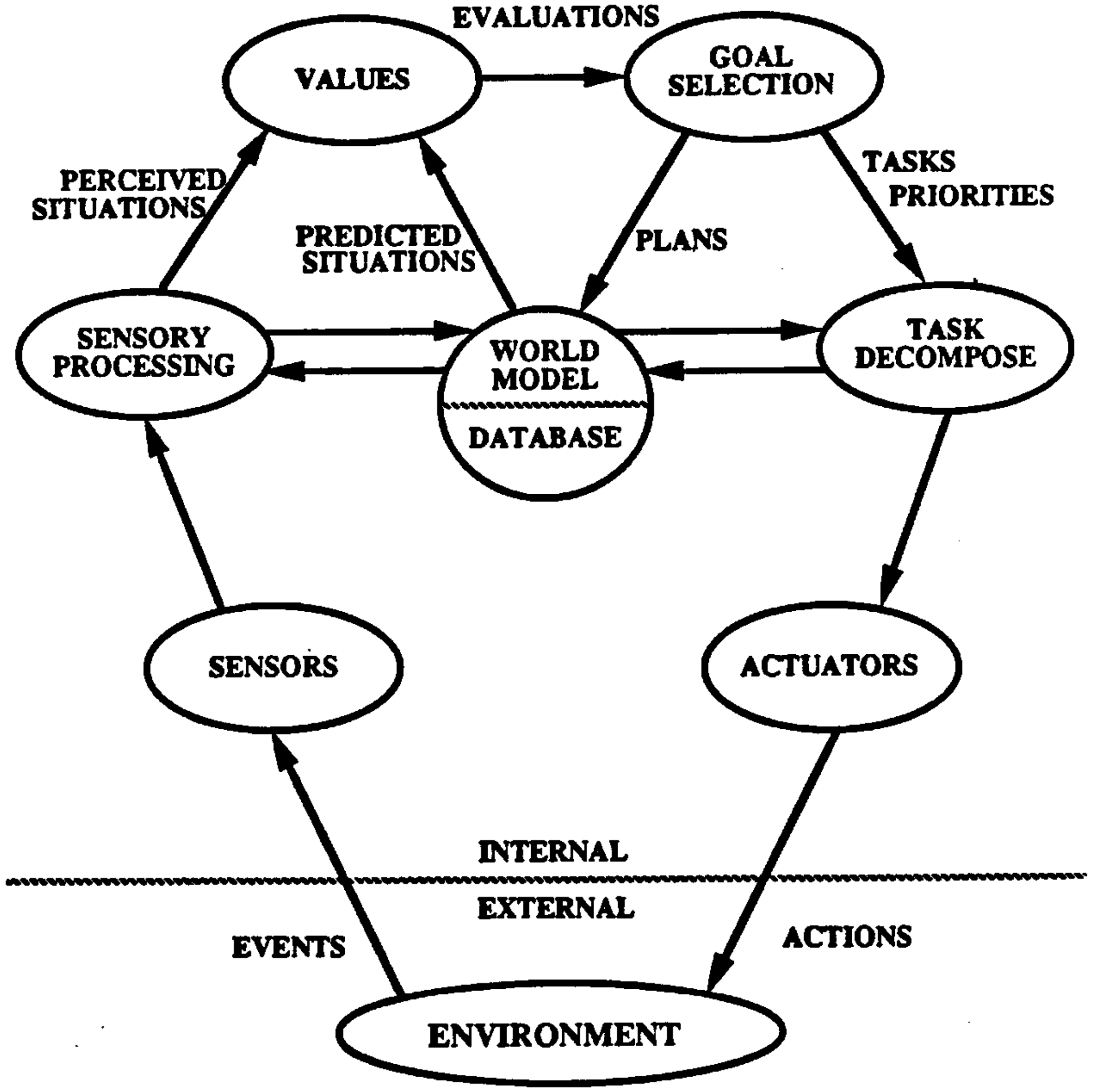


Figure 5.
An intelligent control system

highest value as a goal to be pursued. By this process, the goal selection system chooses goals, generates plans, computes priorities, and assigns resources to tasks so as to maximize benefit and minimize cost and risk.

8. SYSTEM ARCHITECTURE -- An intelligent machine requires an interconnecting system architecture that enables the various components to interact and communicate with each other in an intimate and sophisticated way. A system architecture is what enables the task decomposition system to direct sensors, to focus sensory processing algorithms on objects and events worthy of attention, and ignore things that are not important to current goals and task priorities. It is what enables the world model to answer queries from task decomposition modules, and make predictions and receive updates from sensory processing modules. It is what conveys value judgements from the value estimating system to the goal selection system as to the success of behavior and the desirability of states of the world.

3. The MAUV Control System Architecture

The MAUV control system architecture is hierarchically structured into six levels and is shown in Figure 6 [1,2]. This control system is based on the one developed for the Automated Manufacturing Research Facility at NIST [10]. It is divided into three main components, shown as columns in Figure 6. These are sensory processing, world modeling, and task decomposition. The hierarchy is serviced by a communications system and a distributed common memory. The task decomposition modules perform real-time decomposition of task goals by means of real-time planning, execution and task monitoring. The sensory processing modules detect and recognize patterns, events and objects, and filter and integrate sensory information over space and time. The world modeling modules perform the following functions: (a) they maintain a central real-time database of information about the state of the world and the internal state of the system, (b) they update this database with information from sensory processing, (c) they provide expectations of incoming sensory data, (d) they respond to queries from the task decomposition component based on information in the database and on evaluations of possible future states of the world.

In the task decomposition hierarchy, the highest level, the *mission* level, converts a commanded mission into commands to each of a set of groups of vehicles. These commands involve tasks that treat a whole group of vehicles as a single unit. The *group* level converts group commands into commands to each of the vehicles in the group. These commands involve large tasks for each vehicle. The *vehicle task* level converts task commands into elemental moves and actions for the vehicle. The *e-move* (elemental move) level converts elemental moves and actions into intermediate poses. These are converted into smooth trajectory positions, velocities, and accelerations by the *primitive* level. Finally, the *servo* level converts these into signals to actuators, transducers, etc.

4. Hierarchical Planning and Execution

Before describing the elements of hierarchical planning and execution, we will provide our working definition of a plan, and describe the difference between planning and execution. A plan is made up of actions and events. The events are either events in the world or events in the internal state of the system. We represent a plan as a graph (Figure 7). The nodes of the graph represent actions and the arcs represent events. The purpose of the planner is to obtain a plan graph. It can either generate it or retrieve it from a database.

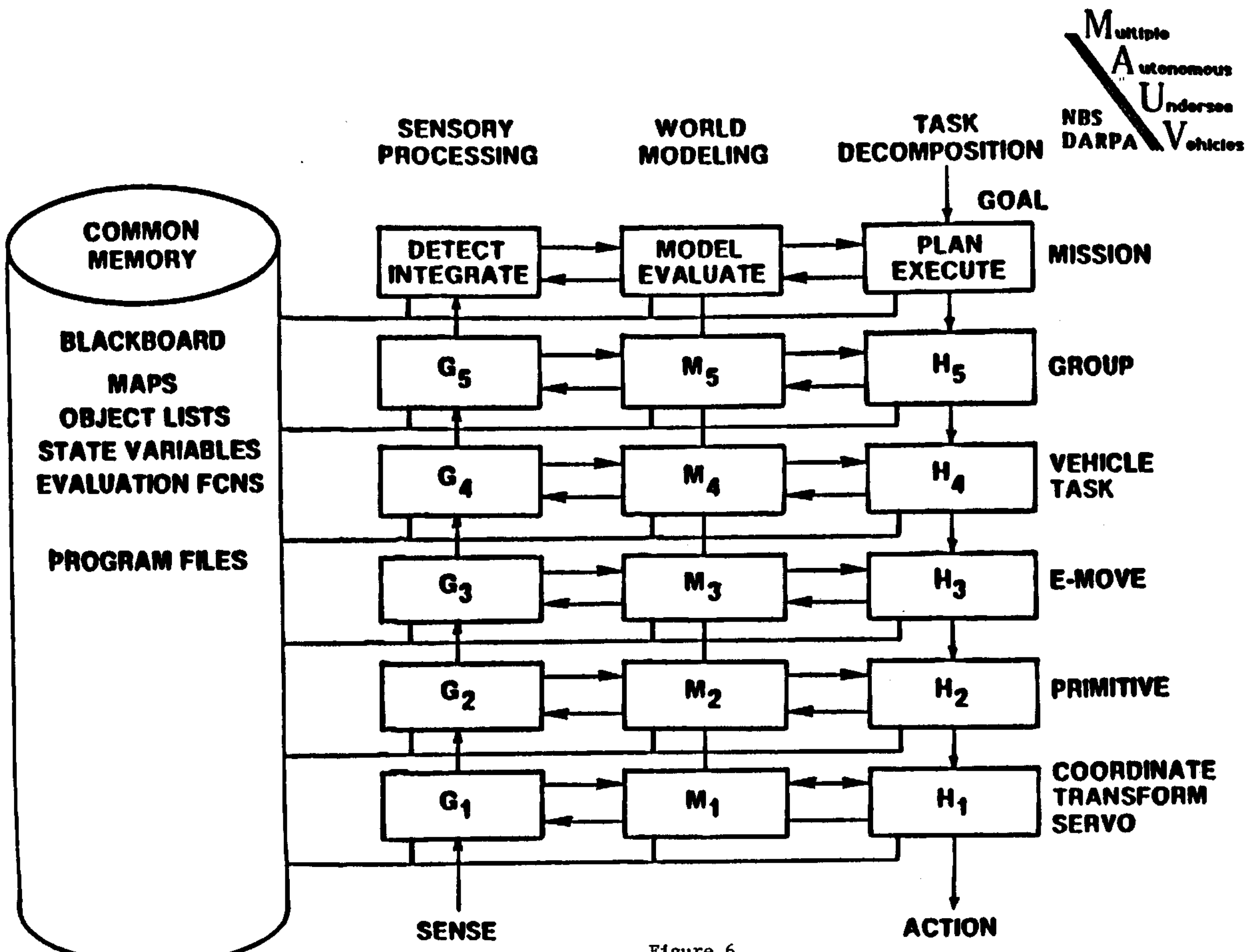


Figure 6.
Block diagram of the MAUV control system architecture.

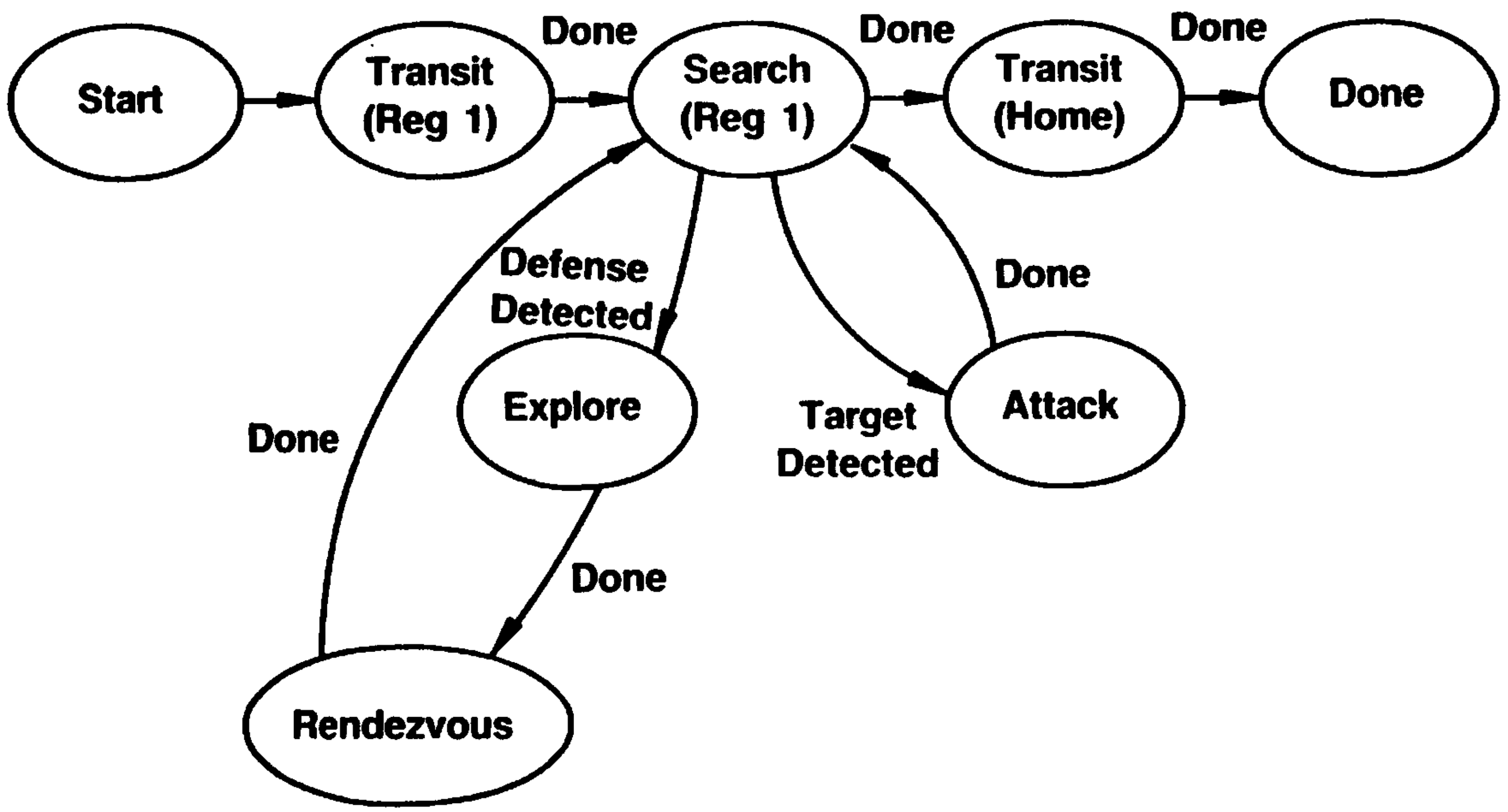


Figure 7.
A plan graph for a mission level plan

We define execution as the process of carrying out a plan. The purpose of the executor is therefore to step through the plan graph. When the executor arrives at a node of the plan graph, it "executes" the action associated with the node. If an action is at the lowest level of the hierarchy, then executing it involves sending signals to hardware. Otherwise, executing an action involves sending it to a lower level where it can be decomposed. As the executor sits at a node of the plan graph, it monitors for events associated with arcs leading out of the node. This monitoring is done at a fast cycle rate. The process of monitoring for an event consists of querying the world model database for that event. If an event has occurred, the executor follows the arc corresponding to that event and steps to the next action.

The notion of hierarchical planning is shown in Figure 8. An action is first input to the top level as a task command. This task is decomposed both spatially and temporally. Spatial decomposition means dividing a task into logically distinct jobs for distinct subsystems. For example, the group level will have a different planner for each vehicle in the group. Temporal decomposition means decomposing a task into a sequence of subtasks. The first step in the plan is then the input task to the next lower level, and this, in turn, is decomposed both spatially and temporally. At each successively lower level, the actions become more detailed and fine structured.

Figure 9 shows a single level of this hierarchy in more detail. The input task to this level first goes to the Planner Manager (PM). The Planner Manager performs spatial decomposition by assigning jobs to each of the planners PL_i. The Planner Manager also coordinates planning among these planners. The planners, operating in parallel, generate their respective plans. Associated with each planner is a separate executor, EX_i, which executes the plan. The executors also operate in parallel.

There are two primary reasons for the hierarchical approach -- to achieve real-time planning and control and to achieve understandability and programmability. At the higher levels of the hierarchy, actions are large scale and they take a long time to execute. Therefore, the search space used to generate plans is coarse and covers large space and time. At the lower levels, actions are smaller scale and they take a short time to execute. The search space is therefore fine and covers small space and time. As a result, the search spaces at all levels are small enough so that the search is manageable. Furthermore, all levels run in parallel.

The understandability and programmability comes about because the control system is decomposed into small modules whose functions can be well understood. Furthermore, different factors are taken into account at different levels, i.e., mission requirements, group tasks, vehicle tasks, elemental actions, etc. In this way, when new knowledge is added to the system, the modules in which this knowledge should reside are more apparent.

5. Levels in the MAUV Control Hierarchy

The MAUV task decomposition hierarchy is shown in Figures 10 and 11. Each module in the task decomposition hierarchy receives input commands from one and only one supervisor, and outputs subcommands to a set of subordinate modules at the next level down in the tree. Outputs from the bottom level consist of drive signals to motors, actuators and transducers. Each large box in Figure 11 has three levels of small boxes inside it. The top level box represents the Planner Manager, the middle level set of boxes represent planners, and the lowest level set of boxes represent the executors, one associated with each planner. The output of each executor is a subtask command to the next lower level.

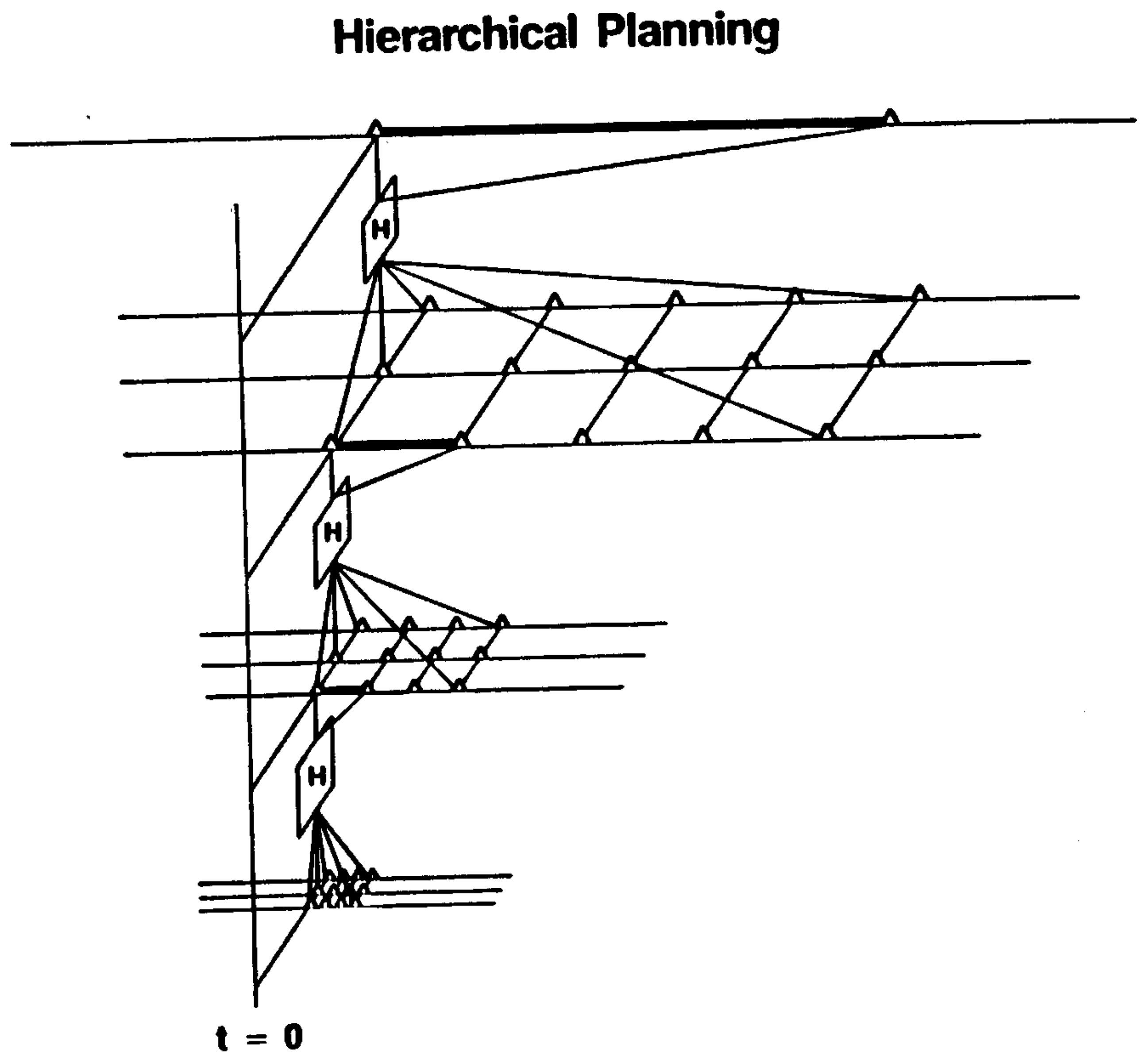


Figure 8.
Three levels of real-time planning activity in the MAUV hierarchy

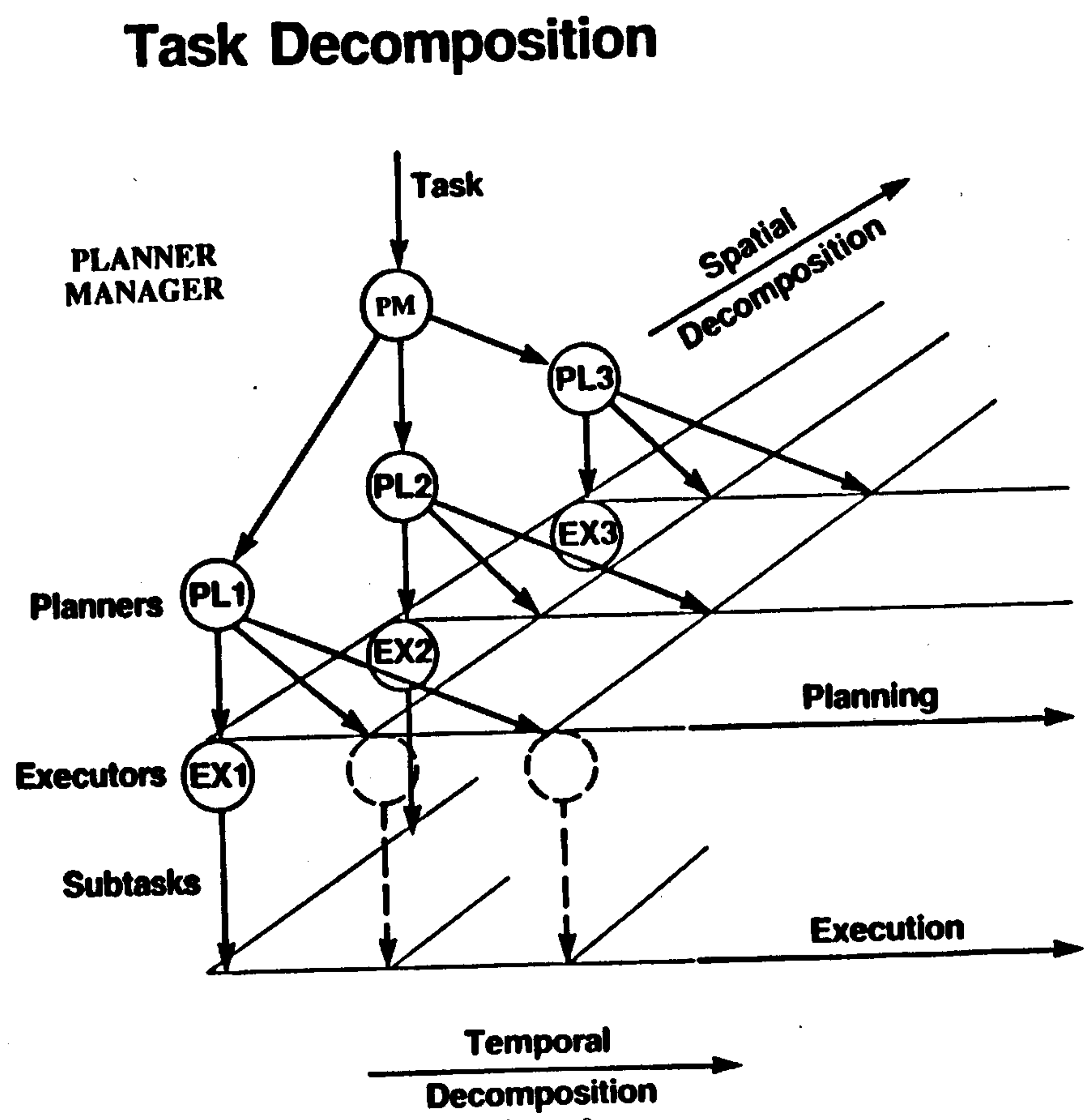


Figure 9.
Internal structure of the task decomposition modules in the MAUV control system architecture at every level of the hierarchy

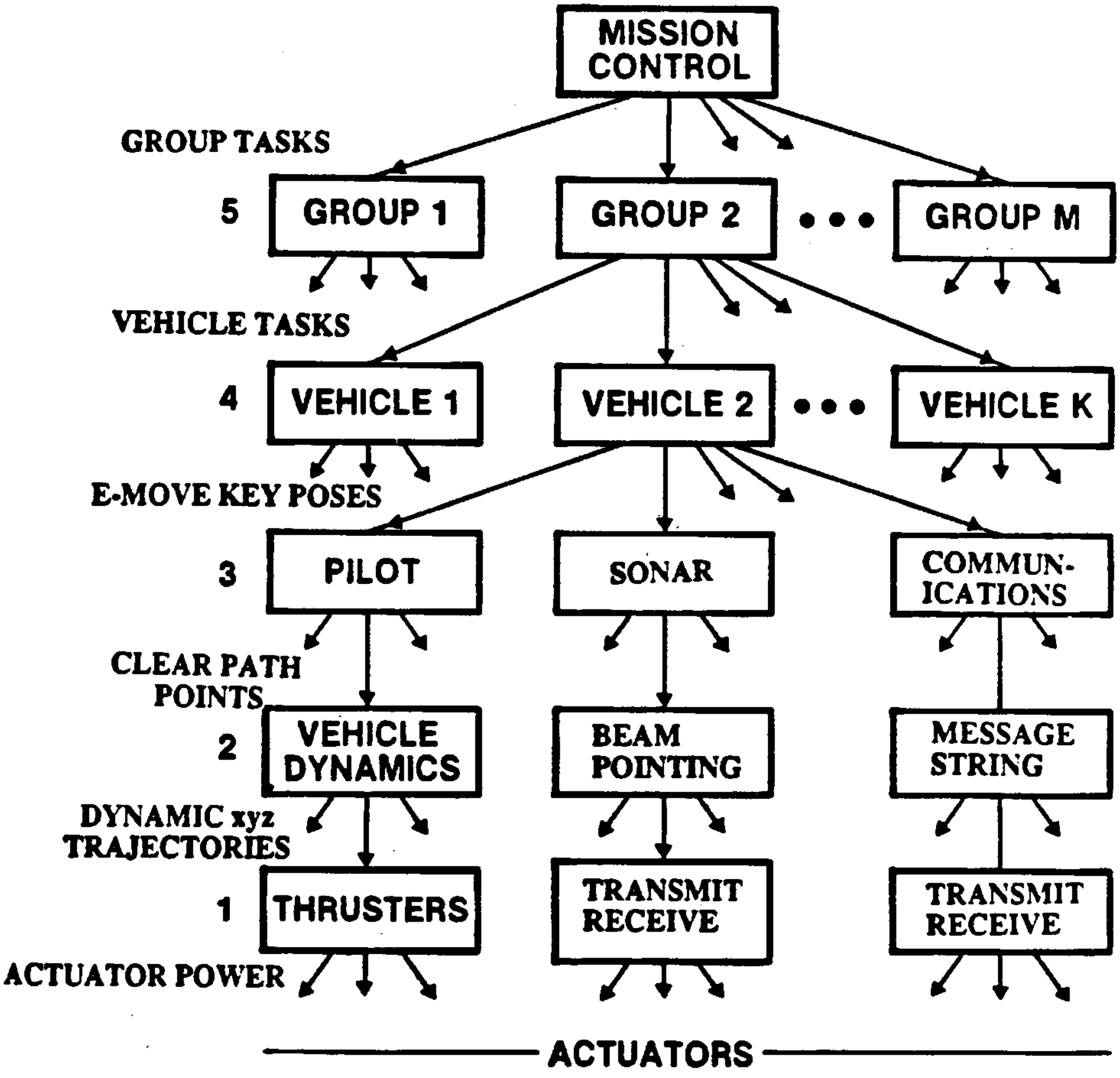


Figure 10.

MAUV task decomposition hierarchy

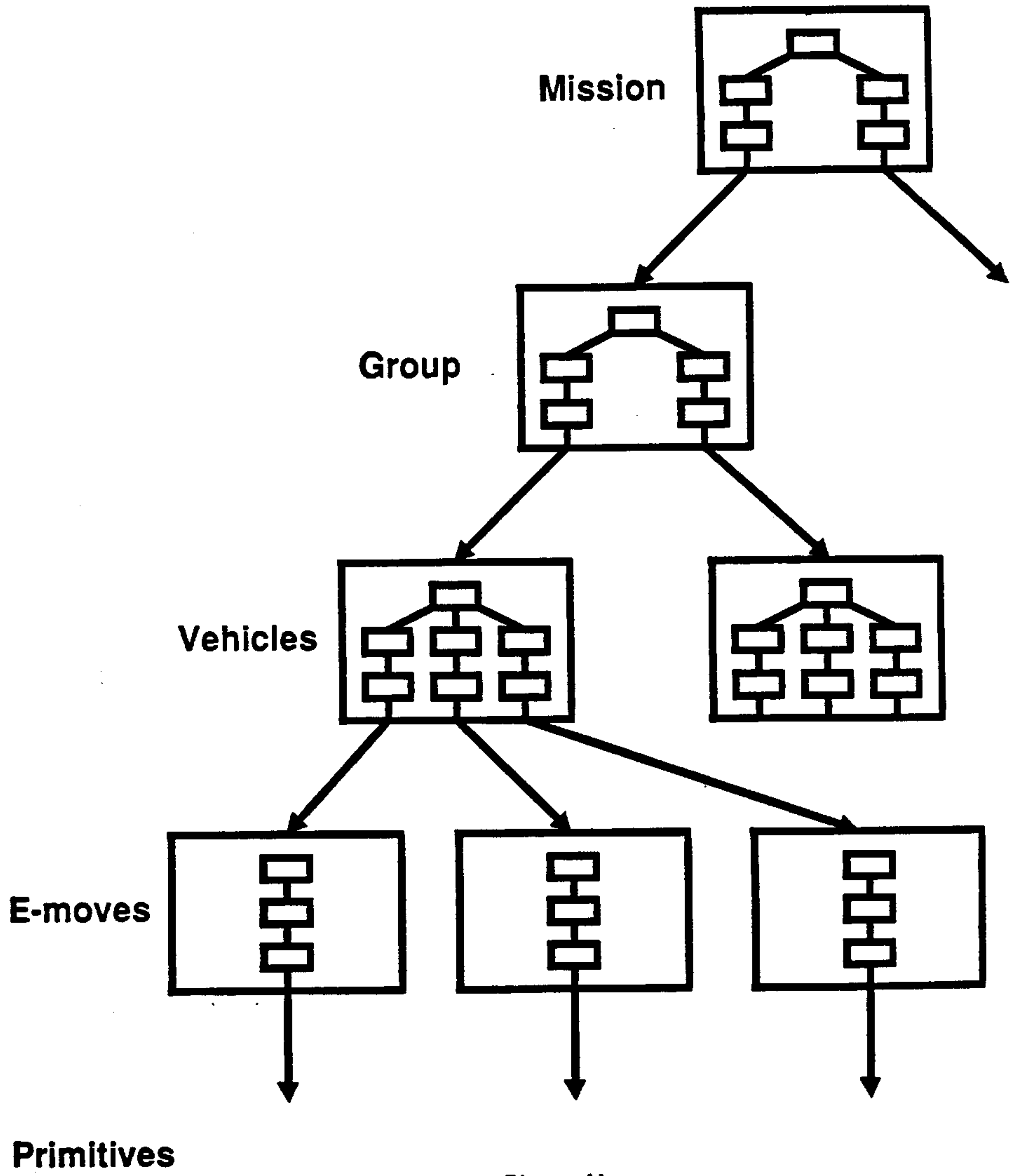


Figure 11.
MAUV task decomposition architecture

5.1. Mission Level

Missions are typically specified by a list of mission objectives, priorities, requirements, and time line constraints. In our implementation, the inputs to the mission level are a command and a mission value function. The command is a task involving a mission strategy, e.g., SEARCH-AND-DESTROY, SEARCH-AND-REPORT, and MAP. Associated with each command is a list of subtasks that define the command. The mission value function is a function used to score the mission, and is composed of the following elements:

1. *A value for each vehicle* -- used to assess the desirability of plan alternatives involving high risk to individual vehicles, or even the deliberate sacrifice of a vehicle.
2. *A value for each subtask* -- specifies the importance of the successful completion of each of the subtasks.
3. *An information value for each subtask* -- specifies the importance of returning information collected while executing each subtask.
4. *A value of stealth for the mission* -- specifies the importance of avoiding detection by the enemy during the mission.
5. *The amount of battery energy* available for the mission.

The function of the mission level is to:

1. Subdivide the vehicles into groups. In our scenario, we have only one group, which contains two vehicles.
2. Determine whether any of the subtasks defining the input mission command should be omitted.
3. Provide a coarse description of routes and tactics for the mission that are sent to the lower levels.
4. Determine appropriate priorities to be used by the lower levels in planning the subtasks.

The outputs of the mission level are the group subtasks and priorities. Piorities are values indicating the importance of the following factors during lower level planning: time used, energy used, stealth, and vehicle survival.

As indicated in Figure 11, the mission level has a Planner Manager, a planner for each group, and an executor for each planner. The Planner Manager assigns vehicles to groups, sets priorities for group actions, and assigns mission objectives to the groups. The planner for each group schedules the activities of the group and sets the priorities mentioned above.

A flow chart for a mission level planner is shown in Figure 12. The program attempts to generate an optimal sequence of subtasks as follows. First, a set of promising plan parameters is chosen. These include a specific sequence of subtasks and an estimate of the time and energy priorities. Next, the planner uses *outcome calculators* to determine the result of choosing these plan parameters. For example, the transit outcome calculator determines the projected risk and the time and energy consumption for each transit leg of the mission. In order to do this, the outcome calculator plans a coarse route. This route will eventually be passed to the lower level planners.

The results of the outcome calculators are then scored based on the mission value function which was input to the mission level. If the score indicates that a clearly satisfactory set of plan parameters has been chosen, then these are passed to the lower level. Otherwise, a new set of plan parameters is chosen and the procedure is repeated. If the time allocated to the planner to make a decision has terminated, the best set of plan parameters thus far found will be passed to the lower level.

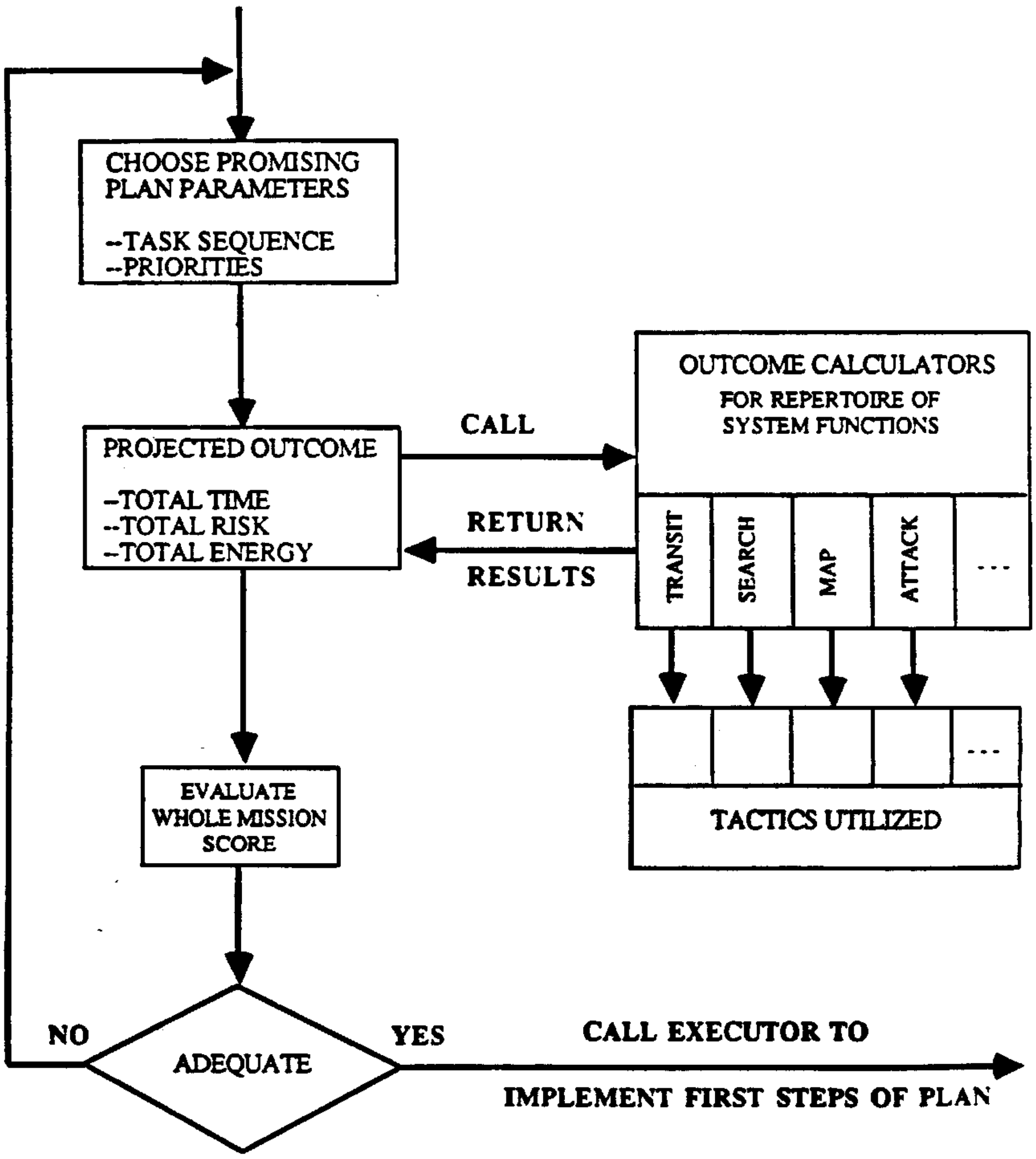


Figure 12.
Mission level planner for a single group

Replanning is done at regular intervals throughout the mission by repeating the program in Figure 12. If replanning results in a different plan from the one currently being executed, it is installed in place of the current plan. In this way, the world and vehicle situation is repeatedly evaluated so that the plan generated from the most recent information is always being executed. Further details about the mission level may be found in [8].

5.2. Group Level

Group task commands define actions to be performed cooperatively by groups of MAUV vehicles on multiple targets. The Planner Manager decomposes group tasks into individual vehicle tasks. This decomposition typically assigns to each vehicle a prioritized list of tasks to be performed on or relative to one or more other vehicles, objects, or targets. Tactics and vehicle assignments are selected to maximize the effectiveness of the group's activity. The actions of each vehicle are coordinated with the other vehicles in the group so as to maximize the effectiveness of the group in accomplishing the group task goal.

Each vehicle planner schedules group task lists into coordinated sequences of vehicle tasks. The vehicle planner uses the group level world model map to compute vehicle trajectories and transit times. They also estimate costs, risks, and benefits of various vehicle tactics (or task sequences).

In our implementation, the inputs to the group level are a command and a set of priorities. The command is a task involving multiple vehicles, e.g., TRANSIT, ATTACK, RASTER-SEARCH. The priorities are values indicating the importance of stealth, destruction, time, and energy. These priorities are used as weights in the cost function during A^* search [5].

In our scenario, there is only one group of vehicles. As indicated in Figure 11, associated with the group is a Planner Manager, a planner for each of the two vehicles in the group, and an executor for each planner.

The planner uses A^* search during planning. The following factors are used in the cost function for this search:

1. *Probability of traversal.* This is based on known obstacles (such as large land masses) and known density of clutter (e.g., a group of small islands in a given path would result in a low probability of traversal).
2. *Probability of detection* by enemy sonobuoy fields or by enemy ships containing acoustic sensors.
3. *Probability of destruction* by enemy minefields or enemy ships containing active sonar sensors.
4. *Energy used.*
5. *Time used.*
6. *Deviation penalty from path specified at level above.* The input task command to the group level may specify a path to be followed. This path is taken into account by the cost function by means of a deviation penalty.

The outputs of the group level are the vehicle tasks and priorities. The output priority values are the same as the input priorities.

5.3. Vehicle Level

The inputs to the vehicle level are a command and a set of priorities. The command is a task performed by a single vehicle, e.g., GOPATH, WAIT, RASTER-SEARCH, LOCALIZE-TARGET, RENDEZVOUS. The priorities are the same as the input priorities to the group level.

The function of the vehicle level is to decompose the input vehicle task into a sequence of tasks for each subsystem of the vehicle. These subsystem tasks are called elemental moves or actions (e-moves). We consider three subsystems, the pilot, sensors and communications subsystems.

As indicated in Figure 11, for each vehicle there is one Planner Manager, three planners (one for each subsystem), and three executors. The Planner Manager decomposes vehicle tasks into work elements to be performed by the various vehicle subsystems. It also coordinates, synchronizes and resolves conflicts between vehicle subsystem plans.

The pilot planner uses the world model database to search for a path between the start and goal positions indicated by the input vehicle command. A^* search is used and its cost function has the same factors as used at the group level.

The communications planner schedules the messages to be sent by deciding if and when to send each message. Currently, this schedule is extracted form a rule database. In the future, the schedule will be determined by computing the value of each message, its urgency, the risk of breaking communications silence, and the power needed to transmit the message.

The sensors planner schedules the activation and deactivation of passive and active sonars. Currently, this schedule is also extracted form a rule database. In the future, the schedule will be determined by computing the value of taking sonar soundings, its urgency, the risk of breaking silence for active sonar, and the power needed to take the sonar soundings.

The outputs of the vehicle level are the e-move tasks.

5.4. E-move Level

The input to the e-move level is a command which is an elemental move or action involving a single subsystem, e.g., GO-STRAIGHT (pilot subsystem), ACTIVATE-ACTIVE-SENSOR (sensor subsystem), SEND-MESSAGE (communications subsystem).

The function of the e-move level is to decompose the input e-move command into a sequence of low-level commands to the particular subsystem controller. As indicated in Figure 11, a Planner Manager, planner, and executor exists for each subsystem of each vehicle.

The pilot e-move can be defined as a smooth motion of the vehicle designed to achieve some position, orientation, or "key-frame pose" in space or time. The pilot planner at this level computes clearance with obstacles sensed by on-board sonar sensors and generates sequences of intermediate poses that define pathways between key-frame poses. A^* search is used to generate these paths. The cost function used during this search uses the following factors:

1. *Traversability*. This is based on known local obstacles. The traversability of a given path is either 1 (the path is traversable) or 0 (the path is not traversable).
2. *Distance travelled.* A shorter path is always preferred. This helps obtain smooth final paths.
3. *Deviation penalty from path specified at level above.* As in previous levels, the input command to the e-move level may specify a path to be followed. This path is taken into account by the cost function by means of a deviation penalty.

A communications e-move is a message. The communications planner at this level encodes messages into strings of symbols, adds redundancy for error detection and correction, and formats the symbols for transmission.

The sensors e-move is a command to activate or deactivate a passive or active sonar. The sensors planner at this level decomposes sonar activation commands into a temporal pattern of sonar pings.

The e-move level is the lowest level currently implemented in the MAUV architecture. The outputs of this level are low-level commands to the subsystem controllers of the MAUV vehicles. These controllers were developed by the University of New Hampshire. For the sake of completeness, we next describe the lowest two levels in the MAUV architecture.

5.5. Primitive Level

The primitive level computes inertial dynamics and generates smooth, dynamically efficient trajectory positions, velocities and accelerations. Inputs to this level consist of intermediate trajectory poses which define a path that has been checked for obstacles and is guaranteed free of collisions.

The outputs of this level consist of evenly spaced trajectory points which define a dynamically efficient movement.

5.6. Servo Level

The servo level transforms coordinates from a vehicle coordinate frame into actuator coordinates. This level also servos thruster direction and actuator power. There is a planner and executor at this level for every motor and actuator in the vehicle.

Inputs to this level consist of commanded positions, velocities, thrust, power, orientation, and rotation rates of the vehicle. Outputs of this level consist of electrical voltages or currents to motors and actuators.

6. Cooperative Vehicle Behavior

Cooperative behavior between the two MAUV vehicles is achieved as follows. The vehicles start out with identical software, except for the vehicle identifier, which is unique for each vehicle. This implies that each vehicle has a mission and a group level, and mission and group level planning is done on both vehicles. If the two vehicles sense the exact same world all the time (i.e., they receive the same sensor input), then mission and group planning will be identical between the two vehicles, and they will achieve coordinated behavior. This is because the two vehicles will generate identical plans for both vehicle 1 and vehicle 2, and each vehicle will simply execute the appropriate plan for itself.

If, instead of always having identical world model databases, the vehicles have the same world model information with regard to significant world properties (i.e., properties relevant to generating and executing mission and group level plans), then mission and group planning will still be identical between the two vehicles. This is the method we currently use to achieve cooperative behavior. The significant world properties relevant to our scenarios are the positions of large land masses such as islands, the positions of sonobouy and mine fields, the positions of the two vehicles, and the positions of enemy targets and defenses. Islands, sonobouy fields and mine fields are input at the beginning of the mission and do not change. Therefore information about these will be identical in the vehicles' world model databases. In order to ensure that information about the other significant world properties are the same in both databases, each vehicle, upon detecting a new target or defense, immediately communicates this to the other vehicle. In addition, each vehicle regularly communicates its position to the other vehicle.

A problem with this technique of achieving cooperative behavior is that, as the scenarios become more complex, more information would have to be regularly communicated between the vehicles. In addition, if a group had many vehicles in it, regular communication from each vehicle to all the others would have to occur. An alternative technique which seems more promising is to designate one vehicle in each group as group leader, and to designate one vehicle as mission leader. The mission leader performs mission planning and communicates the plans to each group leader. Each group leader does group planning and communicates the plans to the individual vehicles in the group. In this way, if different vehicles have different world model databases, they will nevertheless execute cooperative maneuvers determined from the world model databases of the group and mission leaders. If communication cannot occur because of stealth requirements or because a vehicle is out of communication range, then each vehicle still has mission and group level software and can generate its own plans. Of course, this could lead to non-cooperative maneuvers. Once communication is re-established, the mission and group leaders can take over.

7. Real-Time Planning

This section describes the real-time planning system used at the group and vehicle levels of the hierarchy. The block diagram in Figure 13, which shows this planning system, can be applied to the group level as well as the vehicle level. An input task command first goes to the Planner Manager, which contains two modules. The first, the Job Assignment Module, divides the input task into several jobs and sends each to a different planner. The different planners then work on these jobs in parallel. The second module, The Plan Coordination Module, coordinates planning among the various planners. Currently, this coordination is accomplished by generating constraints to be met by all the planners. For example, if each planner corresponds to a separate vehicle, this module might generate constraints consisting of a position where all the vehicles are to rendezvous and a time when this is to occur. Each individual planner would attempt to meet the constraints. If one of them could not, it would report back to the Plan Coordination Module which would then generate a new set of constraints. In the future, the Plan Coordination Module will also coordinate communication among the planners. Some constraints can be determined only by the planners at plan time, and these would have to be communicated to the other planners. For example, one vehicle planner might want as part of its plan one of two actions depending on what another vehicle planner generates.

After a planner has finished generating a plan in the form of a plan graph, the executor associated with the planner steps through the graph.

Each planner contains several modules (Figure 13). The Cyclic Replanning Module accepts an input command (or job) from the Planner Manager and, at regular cycle times, generates a new plan. The primary way in which our system performs replanning is by generating new plans regularly. The traditional way of doing replanning is to post some simple conditions on the world which, when met, causes replanning to occur. Our approach, however, is based on the notion that the best way to know whether the world has changed in such a way as to require a new plan is to actually run the algorithm that generates the plan, and then to see whether the plan has changed. The advantage of doing it this way rather than posting some simple conditions is that there could be a complex interaction of events in the world that would

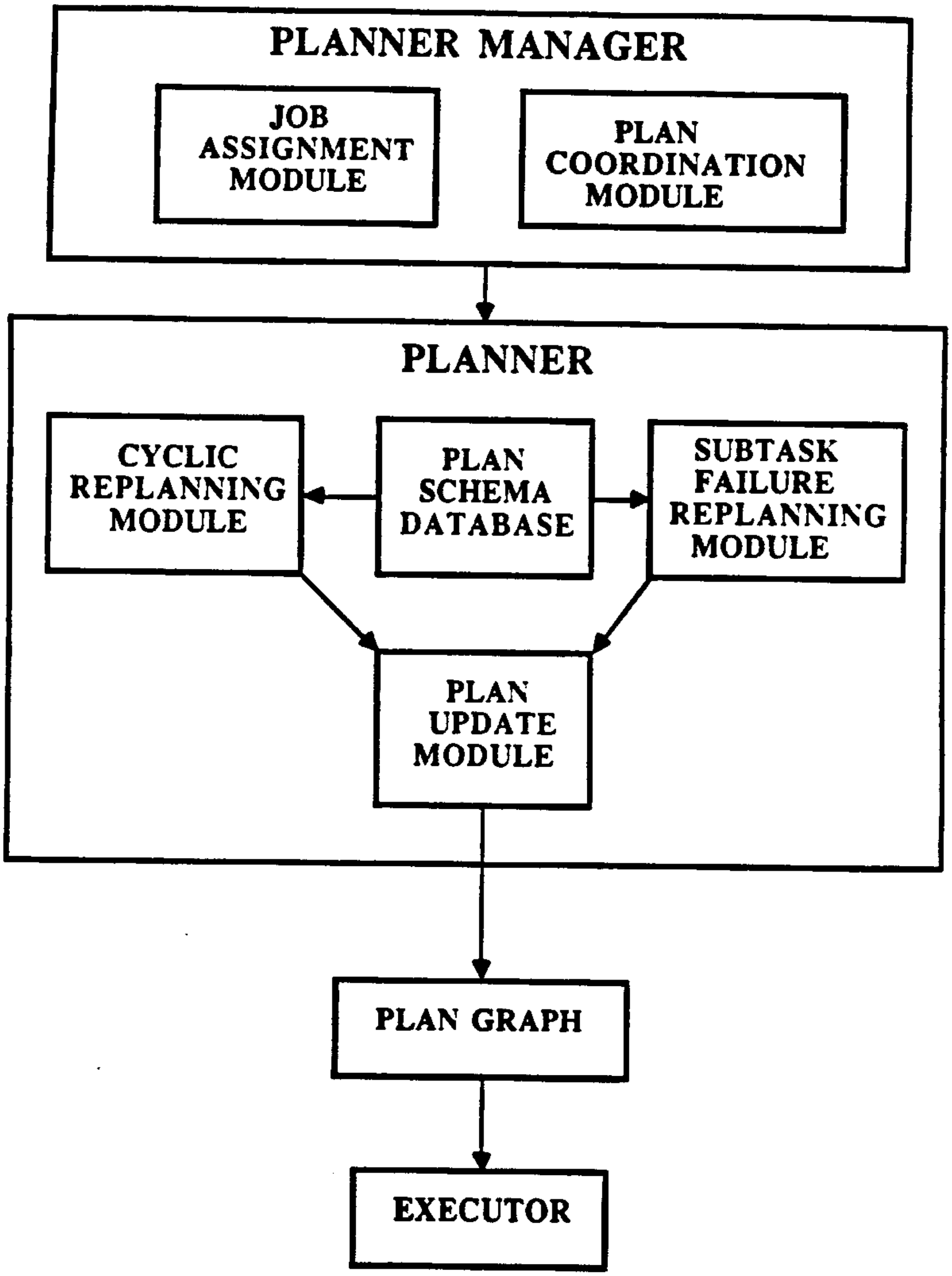


Figure 13.

Internal structure of the planner manager and planners require a new plan, and this complex interaction is exactly what the planning algorithm looks for and evaluates.

One issue that must be considered is real-time planning and how it is handled by the planner. As stated above, we view a plan as being composed of actions and world events. Execution of the plan by the executor occurs by monitoring for world events and stepping to the appropriate action based on which world events have occurred. Let t_1 be an arbitrary point in time and let E be the set of events in the world occurring at t_1. We define *real-time planning* as the process of generating plans quickly enough so that there is always an action a given to the executor such that

1. action a is part of a plan p, and
2. plan p represents an "appropriate" response by the system to events E at time t_1.

Let t_1 be as defined above and let t_2 be the furthermost point in time at which an action must be executed in order to appropriately respond to the world events E. Then the *planning reaction time* is defined as the time interval t_2-t_1.

Fortunately, the planning reaction time is different at different levels of the hierarchy. At the higher levels, the world representation is coarse, planned actions occur over large time scales, and world events are coarsely represented. Therefore the planning reaction time of the system can be relatively slow. At the lower levels, the world representation is detailed, planned actions occur over small time scales, and world events are represented in detail. Therefore the planning reaction time must be fast.

The cyclic replanning time at each level is determined by the planning reaction time. The cyclic replanning times at the higher levels are longer than at the lower levels. At the end of a cyclic replanning time interval, the next action to be taken must be determined by the planner, for the executor must always have an action to carry out. However, these time intervals will often not be enough for the planners to generate new full plans. Therefore, the planner will pass on to the executor whatever is its best plan at the end of the cycle time, even though the planner may not have finished planning to completion. In our implementation, where A^* search is used, the best plan at any point in time is the path in the search tree from the root to the leaf node with lowest cost.

When the Cyclic Replanning Module has generated a new plan, the plan is passed to the Plan Update Module (Figure 13), which updates the Plan Graph.

If a subtask (i.e., an action) of the current plan is sent by the executor to the level below and the subtask cannot be achieved, then a signal is returned to the current level and the plan is modified by the Subtask Failure Replanning Module (Figure 13). Associated with each subtask command sent to the level below is a set of failure constraints. If these constraints cannot be met, then the subtask fails. Examples of failure constraints are (1) achieving the subtask within a time window, (2) achieving a goal (e.g., arriving at a given point in space), and (3) not deviating more than a certain amount from a given path.

The Subtask Failure Replanning Module has thus far been implemented only at the e-move level to handle imminent collision between the vehicle and the lake bottom. The module generates a plan in which the vehicle slowly moves upward, collecting sensory information, until it has determined that there is room to continue forward.

Both the Cyclic Replanning and the Subtask Failure Replanning Modules tap into the Plan Schema Database to generate plans. Plan schemas are used to define the input task commands and will be described next.

7.1. Plan Schemas

A plan schema is used to define a subtask command. It provides all possible sequences of actions that define the command. In order to determine the best sequence in a given situation, it allows the application of a cost function and provides the ability to perform a search which is driven by the plan schema. As shown in Figure 14, the plan schema is represented as a graph. The nodes of the graph represent actions and the arcs represent events in the world or internal events in the system. The plan schema is converted into a specific plan by an interpreter which steps through the plan schema graph and outputs a plan graph. When the interpreter reaches a node in the plan schema graph, it adds the action associated with the node to the output plan. It then queries the world model about the world events associated with the arcs leading out of the node. The queries relate to a hypothetical future world formed by starting with the current model of the world and simulating all the hypothetical actions in the output plan. The interpreter follows the arc whose world event is true, and then processes the next node in the plan schema.

The node of the plan schema is divided into two components, the *alternative action* component and the *context subroutine* component. The alternative action component contains a function that generates all possible alternative actions that can be considered when the node is reached. These alternative actions represent the possible operators that can be applied to the state space at a given point in the state space search. In Figure 14, for example, the GO-STRAIGHT node contains a function that returns all permissible directions for a GO-STRAIGHT action. Since the state space in this case is a three-dimensional grid, all GO-STRAIGHT actions, when executed, will lead to some adjacent point on the grid.

The context subroutine component of a plan schema node contains a subroutine that sets the context (i.e., sets certain variables) for the alternative action component. This context is also assumed for all future nodes of the plan schema that will be traversed by the interpreter. These future nodes also have their own context subroutine components which may modify the assumed context.

The plan schema contains two types of arcs. The first type is a *world event* arc. This arc contains a predicate that queries the world model about a hypothetical future world. A function is then applied to the result of this query, and the predicate returns true or false depending on the value of the function. In Figure 14, for example, the arc out of the GO-STRAIGHT node labeled "@POINT P" is a predicate that queries a hypothetical future world, resulting from the hypothetical execution of a set of GO-STRAIGHTs, about whether the vehicle is at point P. If it is, then the interpreter will step to the HOVER node.

The kind of predicate just described is a *plan time* predicate. Also associated with each world event arc is an *execution time* predicate. This is the predicate that is actually placed in the plan graph, and this predicate will query the most current world model at execution time.

The second type of arc in the plan schema is the *else* arc. This arc also contains plan time and execution time predicates. The plan time predicate returns true if the node that it leads out of has been processed and the predicates of all other arcs leading out of the node return false. In Figure 14, for example, there is an else arc and a world event arc leading out of the GO-STRAIGHT node. If the node has been processed and the predicate of the world event arc (i.e., whether the vehicle is at point P) returns false, then the predicate of the else arc will return true and the node will be revisited. The execution time predicate of the else arc returns true if the node that it leads out of in the plan graph has successfully completed execution and the predicates of other arcs leading out of the node return false.

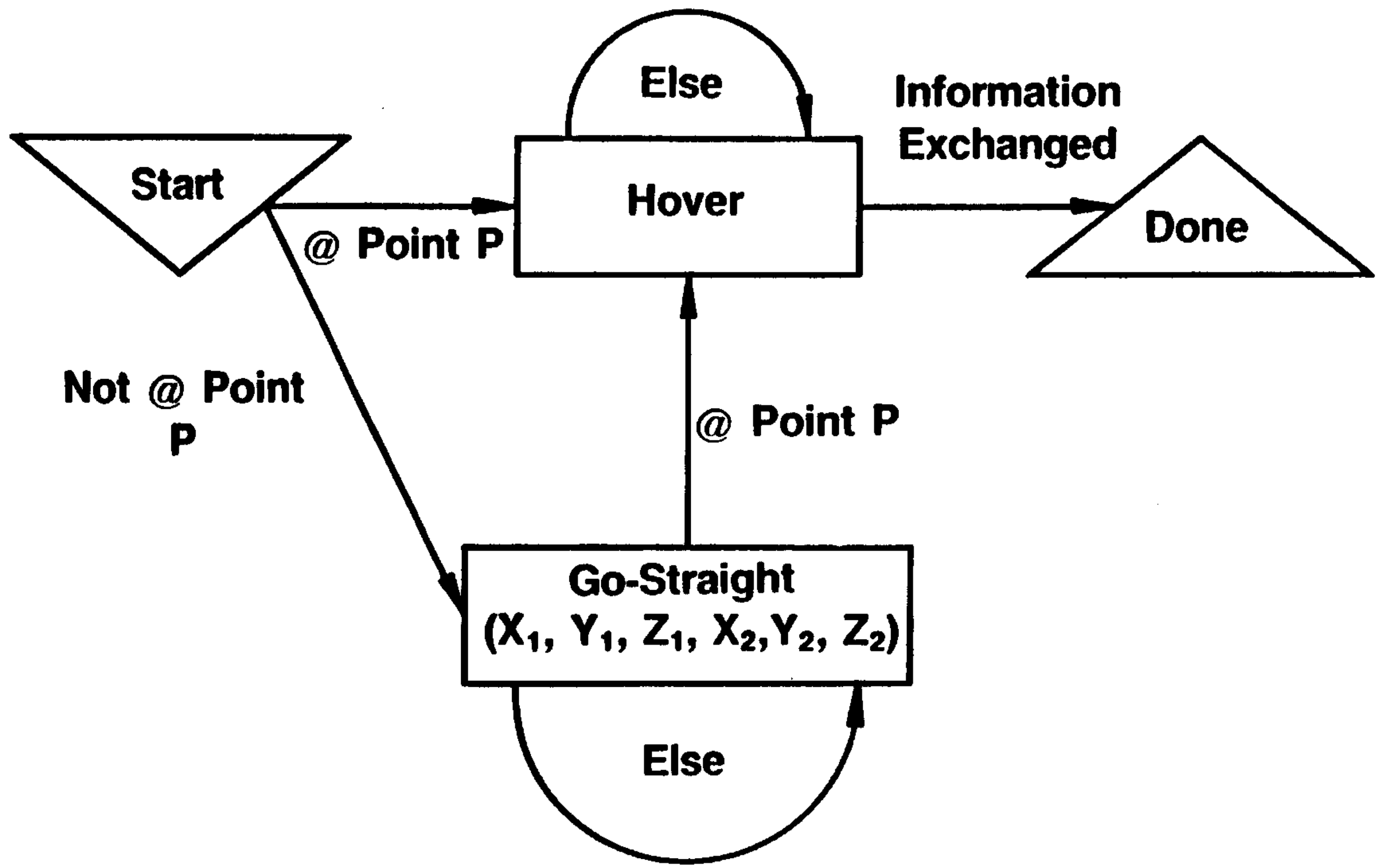


Figure 14.
Vehicle level plan schema for "Rendezvous at point P."

8. World Modeling

The world modeling component serves to accumulate and store information obtained from sensory processing, and to make this information available to the planners and executors. The executors query the world model about the current state of the world so that they can monitor the execution of plans. The planners query the world model about the current state of the world and about hypothetical future states of the world. The world model also provides expectations and predictions to sensory processing.

The world model database is updated from sensory data obtained from the following sensors:

1. Obstacle avoidance sonars -- provide range to obstacles ahead of the vehicle.
2. Flux-gate compass -- provides vehicle orientation.
3. Navigation sonars -- provides vehicle x,y position.
4. Altitude sonar -- provides altitude of vehicle above bottom.
5. Depth sonar -- provides depth of vehicle beneath surface of water (vehicle z position).
6. Pressure sensor -- provides depth of vehicle beneath surface of water (vehicle z position).
7. Defense and target locating sonar -- provides x,y positions of defenses and targets.

In this section, we focus on representing and maintaining the bottom terrain map, with an emphasis on confidence-based mapping in an underwater environment from a sequence of data acquired by six sonar sensors (five forward-looking obstacle avoidance sonars and one downward-looking depth sonar). As the vehicle moves, the information gained from the sonars is used to build an understanding of the environment. Each sonar reading is modeled as a cone, and the positions of the sonar sensors are assumed to be known.

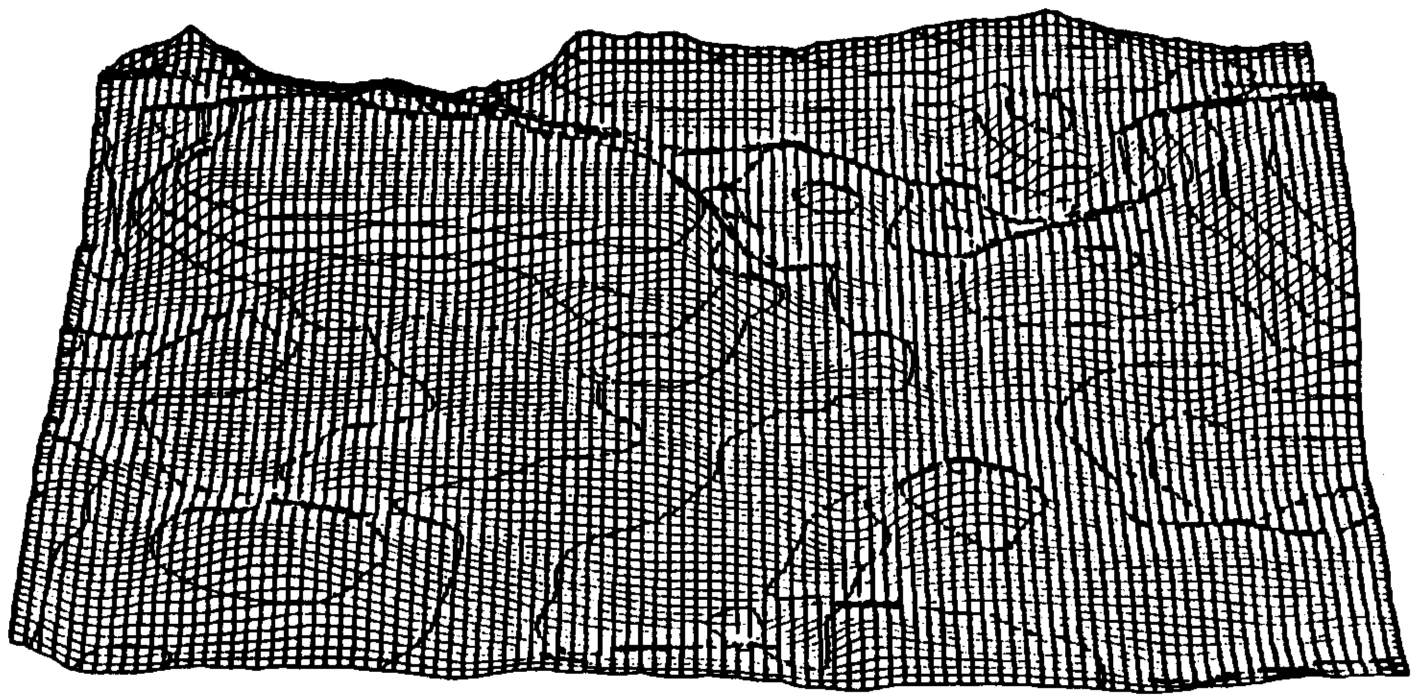
The world model has two types of data for its mapping scheme: a set of global maps, each of which contains data for the vehicle's operational domain, and region-of-interest maps, which only store a localized area around the vehicle's current location. The global maps include underwater terrain elevation data and several overlay feature maps which include data on soil, vegetation, ridges, ravines, landmarks, obstacles, defense points, and transponders. The local maps include terrain elevation and its statistical features.

The *region* quadtree [9] is used to represent terrain elevation in the global maps. The advantages of using a quadtree are that large uniform areas in the map can be described compactly by a small number of large quadrants and that information retrieval is fast since the number of levels in the quadtree is related logarithmically to the resolution of the tree. In addition to the quadtrees used to represent elevation, point and line storing quadtrees have been implemented to provide locations of known objects and topographic features of the lake bottom used in high-level planning. Because the local maps are updated every time new sensor data are obtained; we represent them them as grid structures, which can be very efficiently updated.

8.1. Global Maps

The environment for the MAUV project is Lake Winnipesaukee in New Hampshire. A priori data from a survey of the lake bottom were collected and converted to quadtree format. Figure 15 shows the a priori data for the MAUV mission area.

A separate *sensor* quadtree is used to store higher resolution depth values collected from the sonar sensors during vehicle runs. Both downward- and forward-looking sonars are used in refining the sensor map. A third quadtree stores a depth confidence value for each node in the tree. The confidence map supports the function of distinguishing spurious sonar readings caused by debris or signal inconsistencies from actual obstacles that must be detected and



Lake Winnipesaukee: South, Looking North

Figure 15.

Elevation map of part of the lake bed

avoided. The region quadtree is particularly efficient for sensor and confidence map representation, since unexplored portions of those maps are empty. Such areas can be represented by a small number of nodes in the tree.

Point and line storing quadtrees [9] provide locations of known objects and topographic features of the lake bottom. These simplify tasks such as locating the nearest other vehicle to a given location or plotting a course along linear topographic features like ravines or underwater pipelines.

8.2. Local Maps

Different levels of the control hierarchy require different local map resolutions. Also different types of data may be needed at each level. Generally, the resolution of the map at each level is about an order of magnitude less than the level below. All local maps are implemented as array data structures and only the lowest level (highest resolution) local map updates the global quadtrees. Figure 16 shows the mapping hierarchy for a generalized data set. Arrays are used for their fast, constant access and update time and for ease of implementation. Local maps are generated from the global quadtree database, first by extracting a priori map data for the region, then overlaying the data stored in the sensor and confidence quadtrees, which are presumed to be more accurate than the lake survey information. In fusing the three sets of data, all three quadtrees are traversed over the local map region. Because the updating algorithm only stores data in the sensor quadtree if the confidence measure is above the level assigned to the a priori data, any node for which there are sensor data uses the sensed value. The local map uses a priori knowledge only if insufficient sensor data have been collected for that node. Confidence quadtree values are also copied into the local map.

In the current implementation, the mission level map divides the area into a coarse grid of approximately 25 x 25 pixels, each pixel storing the average depth of the corresponding area. The next two levels in the hierarchy, the group and vehicle levels respectively, share the same local map for this data set. Each pixel of the local map stores the minimum and maximum known depths over a 4 x 4 meter area. It serves the purpose of providing information for high level navigation tasks, such as determining the probability that an area is traversable by one or more vehicles. This map is updated as new information is added to the lowest level map, the e-move map. The e-move local map has the highest resolution (each grid square represents a 0.5 x 0.5 meter area), and is used in determining the traversability of a path between two specified points. The world model returns a probability that the path is traversable based on the information in this map. For example, the output may be a percentage of pixels for which the vehicle clears the lake bottom over the hypothesized path. In the simplest case, the world model can provide a probability of 1 if all of the pixels are traversable, or 0 if any are obstructed. Typically, the e-move pilot planner will query the world model for the traversability of several paths, using A* search to choose the best path. The e-move map is also the level updated directly by sensor readings; its modifications are propagated up through the mapping hierarchy.

8.3. The Updating Algorithms

At the beginning of a mission, the MAUV control system initializes the global and local maps, reading available a priori knowledge from a secondary storage device. The sensor quadtree is initially composed of a single, empty node, though it could also contain sensor data stored from previous missions if available. Likewise, the confidence quadtree is initialized as a single node containing a base confidence value, unless there is confidence data from a previous

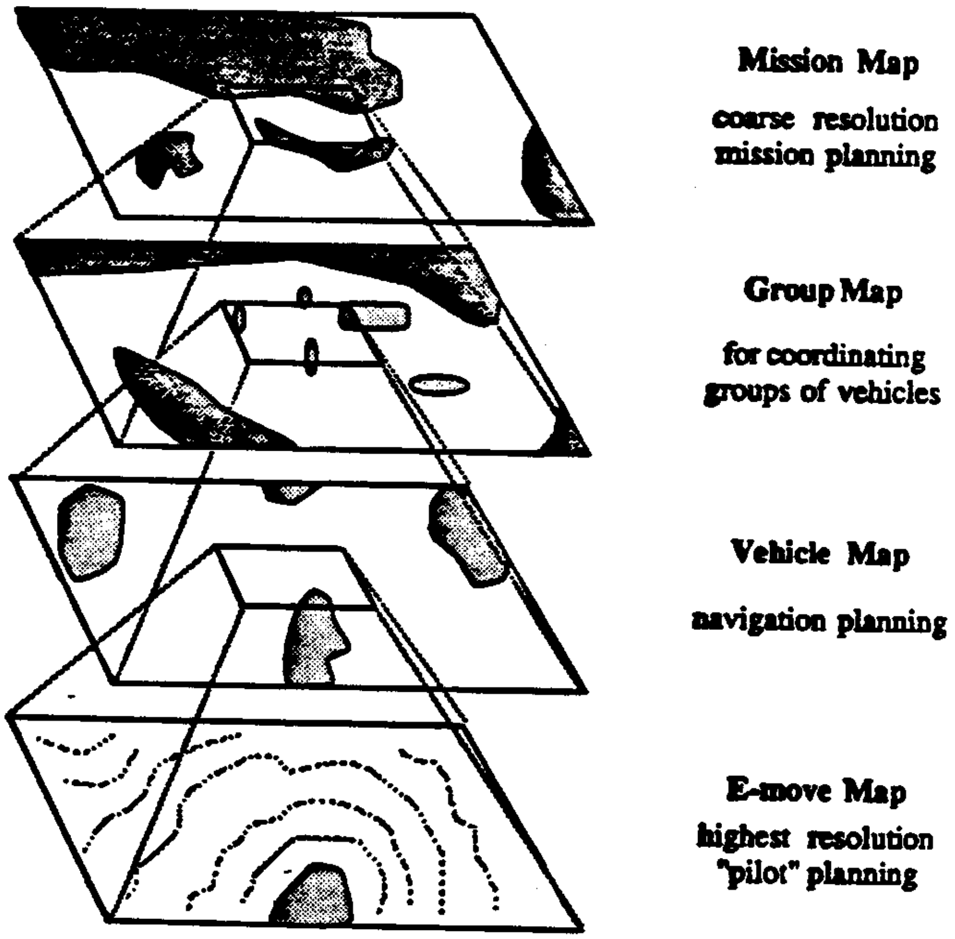


Figure 16.
MAUV mapping hierarchy

mission. In general, the world model starts up in a state of total dependence on a priori knowledge, gradually becoming more reliant on the current sensor map as data are collected.

In updating the map from downward-looking sonar data, the algorithm first computes an approximate neighborhood size of pixels to be updated around the current vehicle location which depends on the width of the sonar beam and the distance to the lake bottom. Given that the beam width is fixed and the range is returned by the sensor, a closed-form trigonometric solution can be performed using a lookup table. Although the 2-D projection would be best represented as a circular region, for our purposes, a square neighborhood is sufficiently accurate and more efficient to update. The depth stored at each pixel of the neighborhood in the local map is compared to the observed sonar reading. If the two values are not within an acceptable margin of error, the conflicting data cause the pixel's confidence to be lowered. If the two depth values are in agreement, the confidence value is incremented unless it has already reached the maximum allowed. Whenever a pixel's confidence value drops below the predefined threshold, it takes on the new depth reading and is assigned a base confidence value (Figure 17). For the depth sonar, all information is classifiable as either conflicting or agreeing with the knowledge already in the model. None of the data are irrelevant in this case.

The obstacle avoidance (forward-looking) sonar mapping algorithm is more complicated. Here the projection of the cone into the two-dimensional plane approximates a triangular region. The cone itself is approximated by two planar surfaces representing the top and bottom surfaces of the cone. Due to the relatively coarse resolution used in the obstacle avoidance algorithm ($0.5m^3$ per pixel) and the narrow width of a sonar beam, this does not introduce significant error into the calculations. As with the depth sonar algorithm, each pixel in the 2-D projection is examined and updated if its confidence value drops below the threshold. Forward-looking sonar readings provide two types of information: a given pixel may be clear, or it may be obstructed by an obstacle. When the vehicle detects an obstacle, the mapping algorithm adds the information to the local map by raising the modeled bottom of the lake at that location (i.e. making it shallower, see Figure 18).

It is also an essential function of the world model to be able to remove hypothesized obstacles in the local map as well as add them. For each pixel in the triangular projection, if the three dimensional distance (measured along the cone trajectory) from the sonar source to the current pixel being examined is less than the range returned by the sensor, the pixel is assumed to be clear. No obstacle was detected there, so the depth at that location in the local map should reflect this information. Its value should be greater than or equal to the depth of the bottom surface of the sonar cone at that location, since any object obstructing the beam would presumably cause the sensor to return the range to that object. If the local map value is shallower than the beam, it conflicts with the new sensor data and the confidence value is decremented. If this results in a confidence lower than the threshold, the pixel is reassigned the depth value of the bottom surface of the cone and a new base confidence value. Note however that a local map value in agreement with sonar information does not necessarily increase its confidence. The sonar beam may be projected in front of the vehicle when it is near the surface, and a clear reading near the surface would not yield any information about the depth of the lake bottom if we already have some a priori knowledge that the lake is approximately N meters deep. In this case it would be considered irrelevant data.

The same is not true for pixels in the projection whose distance from the sonar source is greater than or equal to the range returned by the sensor. These pixels correspond to detected obstacles and the depth values in the local map are compared to the top surface of the cone. Here the local map data should be at least as shallow as the top surface of the beam to be in agreement with the sensor reading. If the map data does agree, it represents confirmation of an existing obstacle and the confidence value should be incremented. It should be noted that this

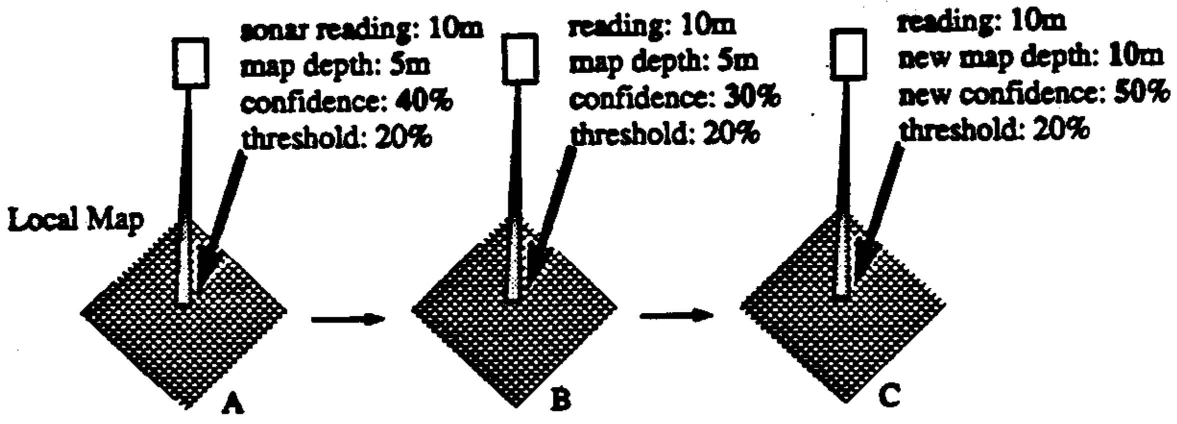


Figure 17.

Three stages of a map update from depth sonar: (A) Incoming sonar data conflicts with the world model. (B) Confidence values for the corresponding world model pixels are decremented (depth is not modified yet). (C) After repeated conflicting readings, confidence drops below the threshold, the depth is updated, and a new confidence assigned.

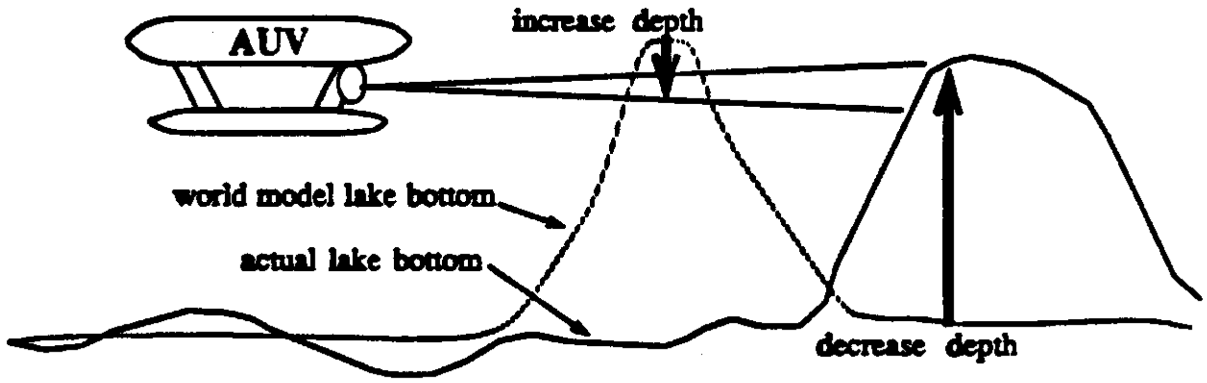


Figure 18.

Updates from obstacle avoidance sonars remove false obstacles in the world model by increasing depth values in the map. Obstacles are added by decreasing the depth, in effect, raising the bottom of the lake model.

confirmation only supports the hypothesis that there is an obstacle at the depth it was detected; no conclusions can be drawn as to the true height of the object or whether it extends all the way to the lake bottom.

In a similar manner, if the model continually disagrees with the sensor reading, the confidence is decremented until the depth value is reassigned to the depth of the top surface of the cone, making the model shallower. Its confidence is again initialized to a base value. Further details about the world model may be found in [6,7].

9. Timing

An important issue for real-time control is timing of processes. In discussing the timing in the MAUV system, we consider the following factors at each level of the hierarchy: executor cycle period, input command update interval, replanning interval, and planning horizon.

The input command update interval is the rate at which new commands are input into a given level from the level above. The replanning interval is how often the planners at a given level do cyclic replanning. The planning horizon is the amount of time into the future covered by a plan at a given level. The executor cycle period at each level is the rate at which the executor checks to see whether a new output command is to be sent to the level below. This cycle period is relatively fast. Table 1 shows these values for each level of the hierarchy.

Table 1: Timing values

Mission Level	Replanning Interval	~30 min
	Planning Horizon	~2 hr
Group Level	Input Command Update Interval	~30 min
	Replanning Interval	~5 min
	Planning Horizon	~50 min
Vehicle Level	Input Command Update Interval	~5 min
	Replanning Interval	~1 min
	Planning Horizon	~10 min
E-move Level	Input Command Update Interval	~1 min
	Replanning Interval	~10 sec
	Planning Horizon	~2 min
Primitive Level	Input Command Update Interval	~10 sec
	Replanning Interval	~2 sec
	Planning Horizon	~20 sec
Servo Level	Input Command Update Interval	2 sec
	Replanning Interval	600 msec
	Planning Horizon	4 sec
	Output Command Update Interval	600 msec

The executor cycle period at each level is the same -- 600 msec. This is the rate at which new sensor data are collected. Therefore, the executor need not cycle faster than this since it will not determine that there can be a new output command unless new information about the world is known. The input command update interval increases by about a factor of five as we go up the hierarchy. The time values given in the table above represent approximate average times. For example, the rate at which new input commands can be received can be as fast as 600 msec (the executor cycle period) at any level. However, we do not expect this to happen very often.

The replanning interval at a given level is the same as the output command update interval at that level. In this way, the planners attempt to replan before each next command is determined.

The planning horizon at a given level is about twice the input command update interval at that level. Each planner therefore generates a plan that represents a decomposition of the current input command as well as the next input command.

10. Implementation

The control system was implemented on the computing systems shown in Figure 19. In each vehicle, a VME bus supports high bandwidth communication between sensory processing, world modeling, planning, and execution modules at each level of the hierarchy. These modules are partitioned among three separate single board Ironics computers so as to maximize the use of parallel computation. A two megabyte common memory board is used for communication between processes, and an 800 megabyte optical disk is used for mass storage. The real-time multi-processor, multi-tasking operating system used is pSOS.

Also shown in Figure 19 is the software development and simulation environment. A variety of computers, including Sun workstations, a VAX 11/785, a micro-VAX, IRIS graphics systems, PCs, Duals, and Ironics development systems are tied into the development environment for code development and simulation. Once the software has been translated to run on the Ironics Unix-based development system, it can be compiled to run under pSOS and downloaded into the 68020 target hardware for real-time execution.

11. Experimental Results

This section describes some intial experimental results on lake tests performed with one of the MAUV vehicles. These tests were performed during October 1987. Due to lack of continued funding, the MAUV project was terminated in December 1987. We were therefore unable to perform all of the demonstration scenarios described in the Introduction.

The lake tests were performed at Lake Winnipesaukee and were run using code at the servo, primitive, and e-move levels. The first experiment involved local obstacle avoidance. Figure 20 shows the path executed by the vehicle during a test run in which an obstacle was manually entered into the world model map at point C, and the vehicle was commanded to go from point A to point B. The control system succesfully planned and executed a path around the obstacle at point C.

The second experiment involved following along a predefined path. Figure 21 shows a raster-scan path from point A to point B. The vehicle determined its x,y position from acoustic navigation transponders which receive signals from navigation bouys placed in the water. The actual path executed by the vehicle during this run is shown in Figure 22. One of the obvious problems brought out by this run is that the vehicle tends to overshoot when it makes turns. This is a problem with the current low level control, which allows position control but not velocity control. Because the velocity is at maximum value when it takes a turn, it will always overshoot. Also, there is considerable error in the position measuring transponders, which largely accounts for the ragged appearance of the pathways.

The third experiment involved updating the internal model of the lake bottom with altitude information obtained from the downward looking depth sonar. Figure 23 shows three graphs. The top and middle graphs display the x and y positions, respectively, of the vehicle

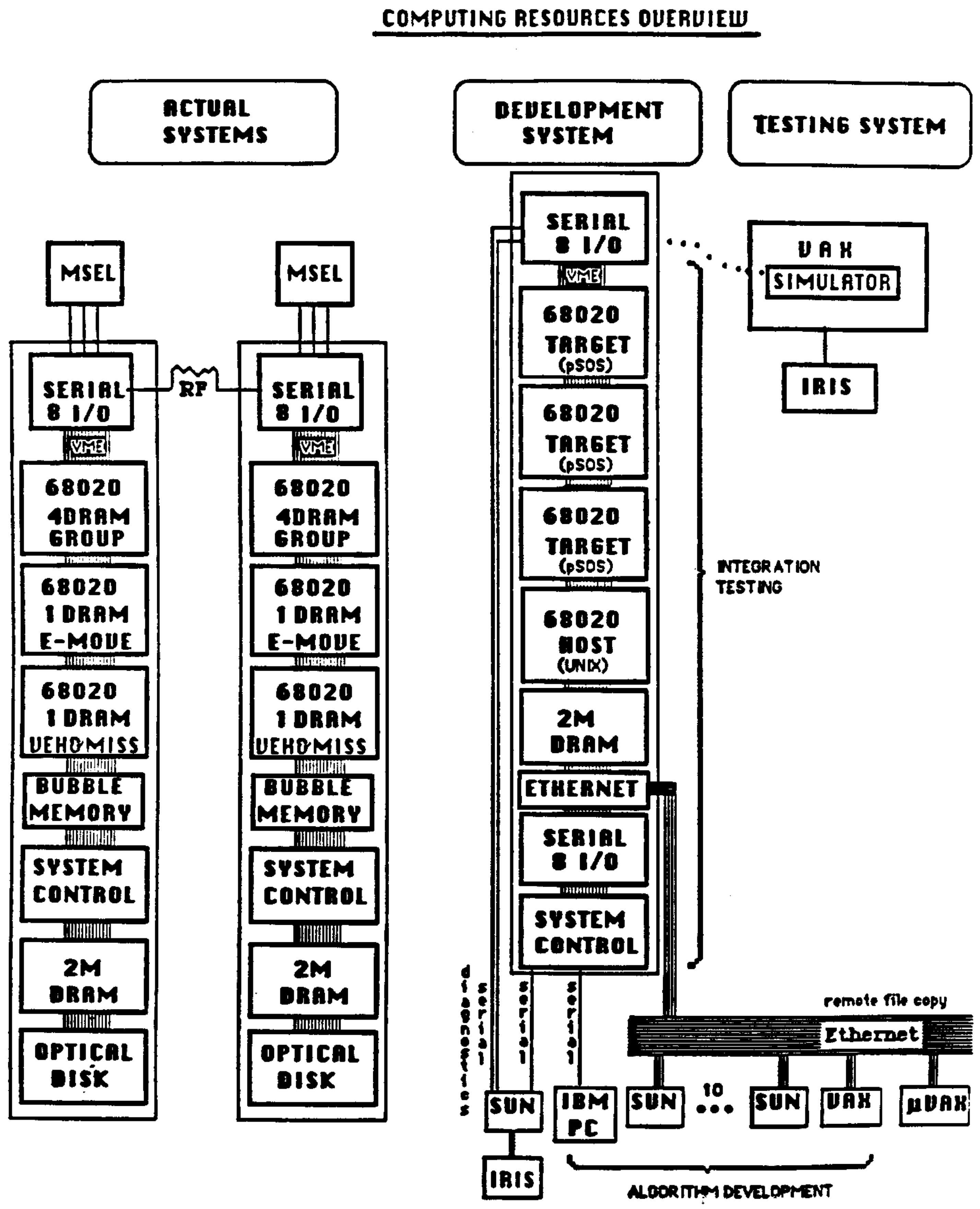


Figure 19.

On the left is the target hardware for the two MAUV vehicles. On the right is the MAUV software development and simulation environment.

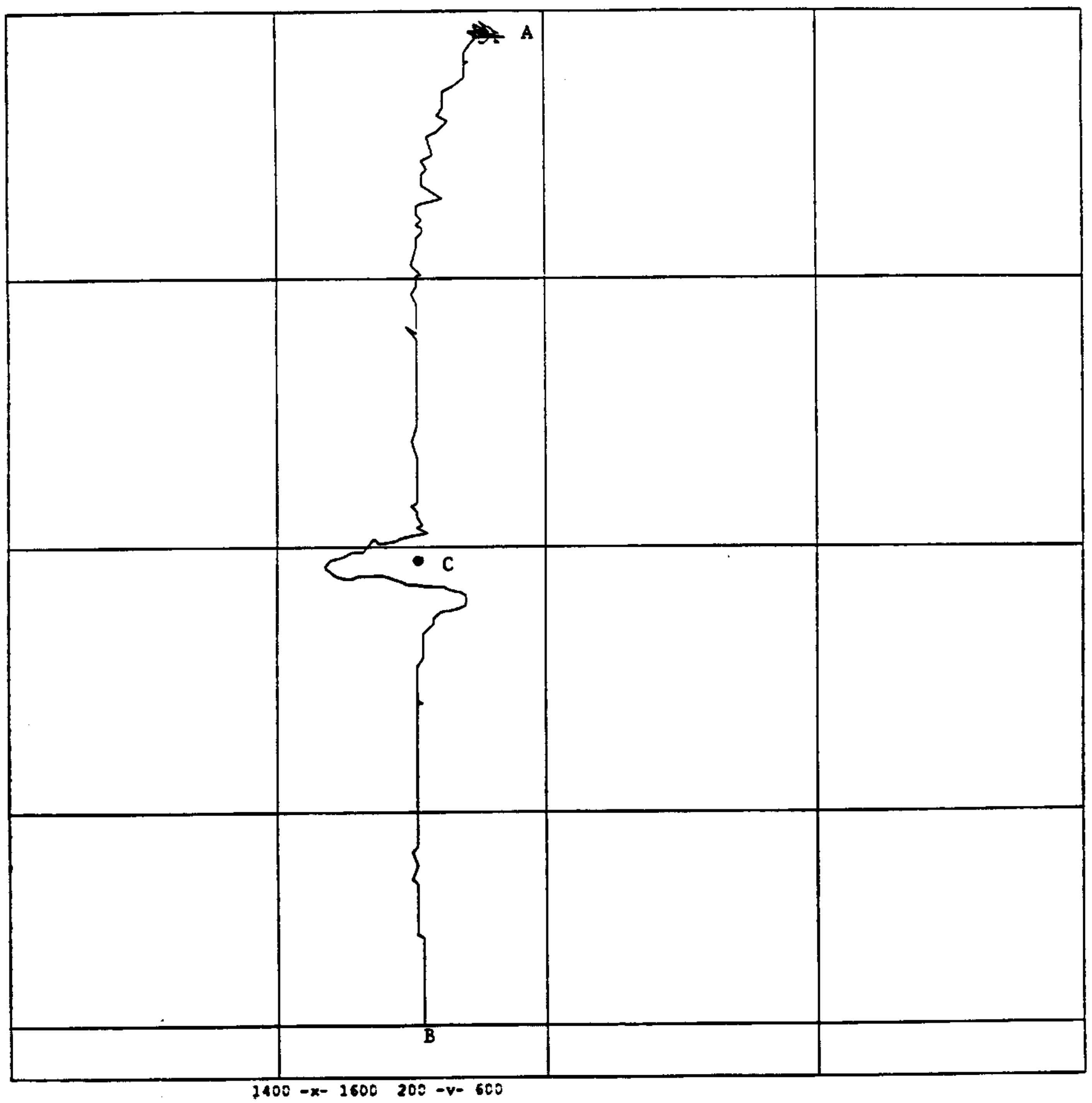


Figure 20.

Obstacle avoidance test run

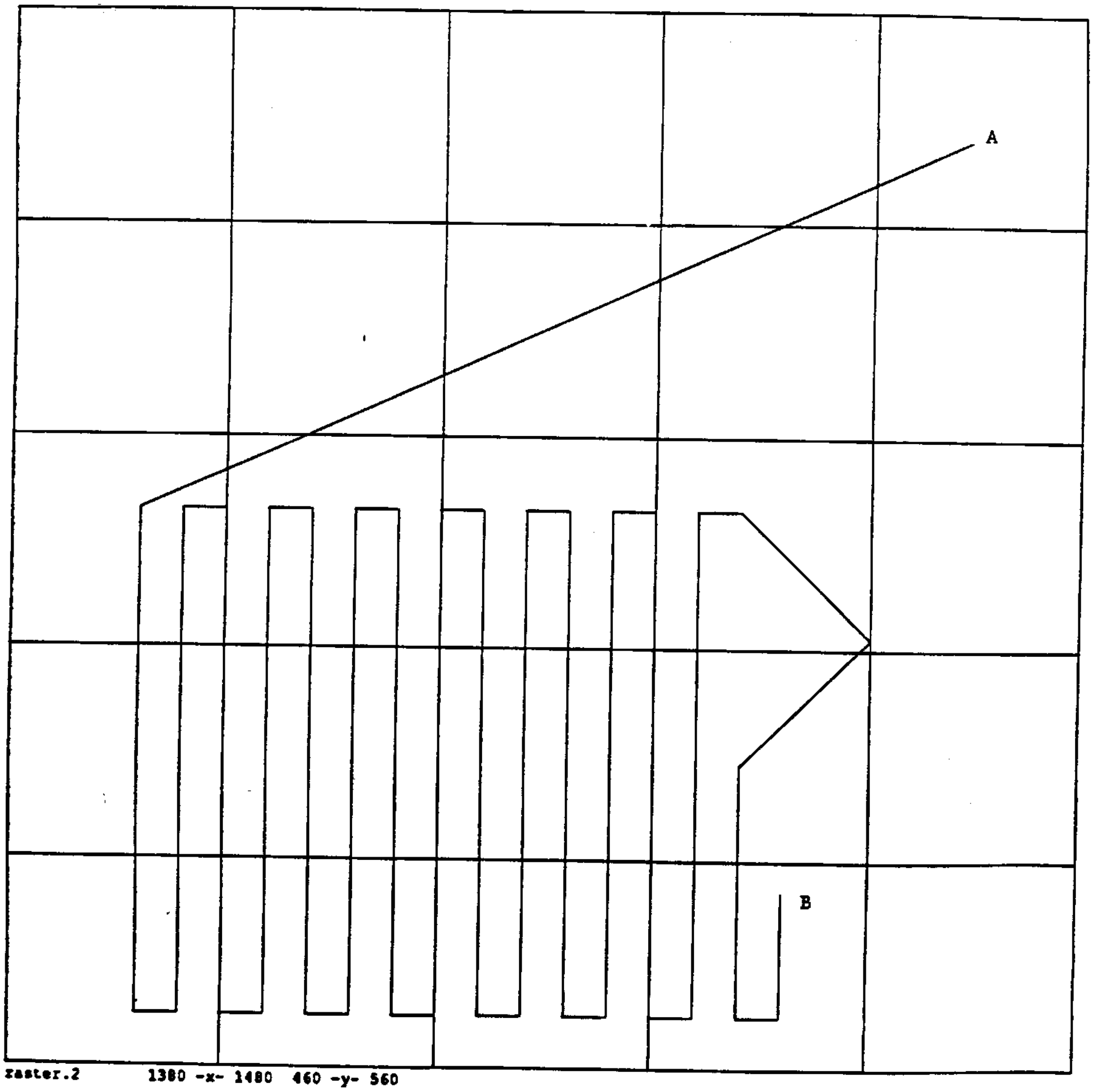


Figure 21.

Predefined raster scan path

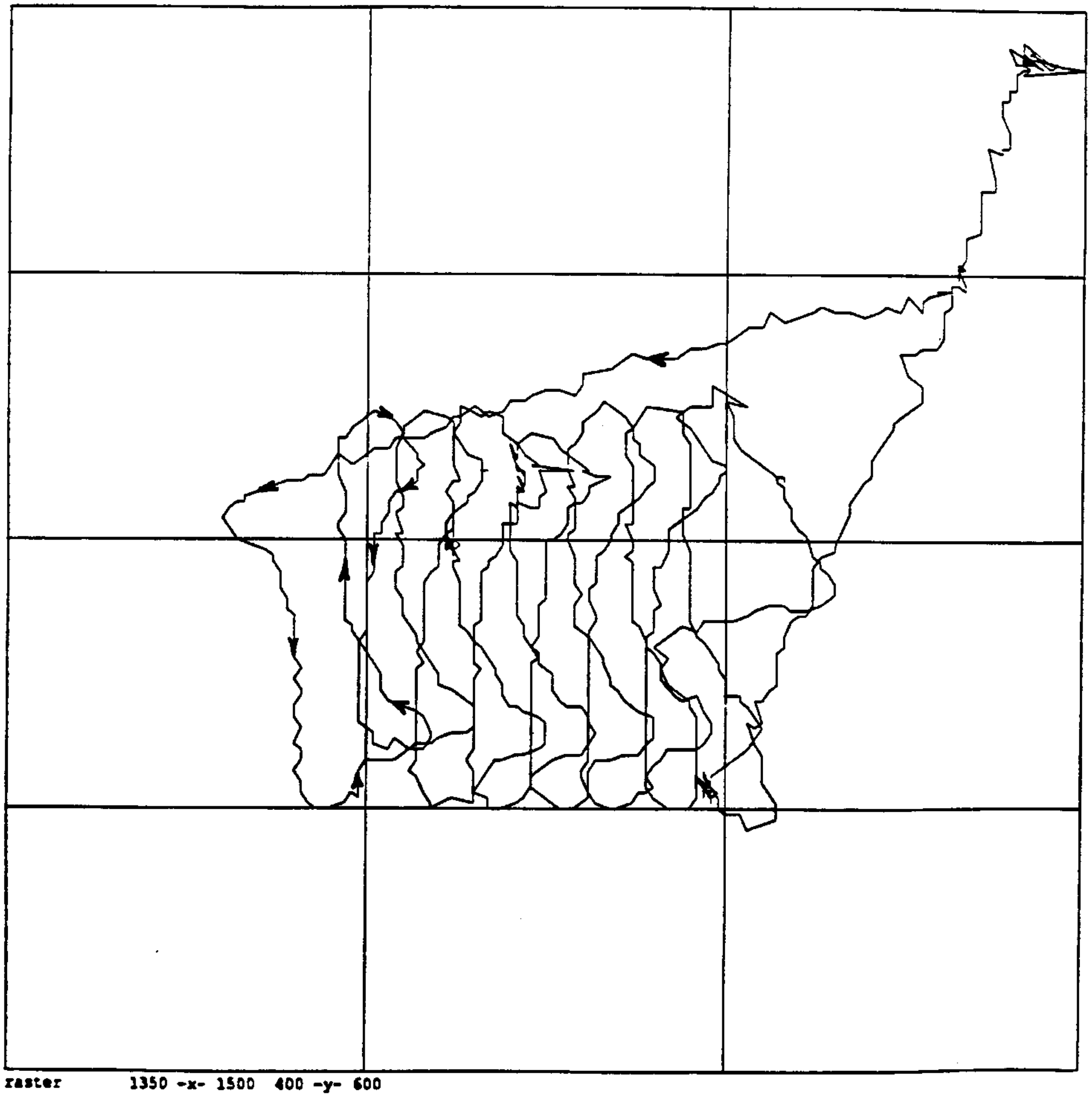


Figure 22.

Actual raster scan path executed. Arrows show direction traveled.

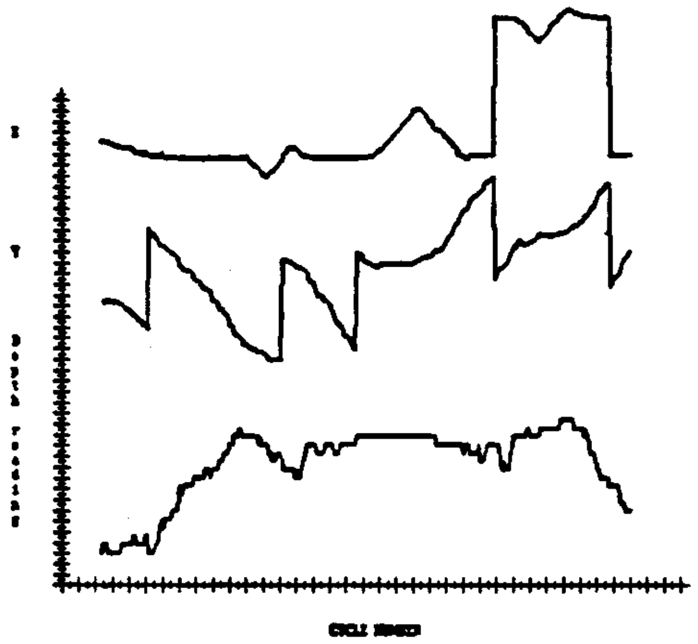


Figure 23.

Updating the world model lake depth

path. The bottom graph shows the lake depth values obtained from the world model along this path after the world model is updated from the information in the depth sonar.

12. Conclusion

The achievement of real-time intelligent control for autonomous vehicles will require a system that integrates artificial intelligence with modern control theory, and that can be implemented on parallel, possibly special-purpose hardware. This paper has presented the basic components of such a system, and has presented a hierarchical control system architecture that can serve to integrate the various components. A first cut at the algorithms and software for many of these components has been developed and is presented here. However, many of these algorithms need to be improved to handle more complex scenarios.

A major problem is the achievement of real-time performance. Although the multiprocessor computing system described here can serve as a good basis for achieving real-time performance, we believe that it must be augmented with special-purpose hardware such as real-time sensory processing devices (e.g., PIPE [4]) and massively parallel devices (e.g., neural networks). Generally, such special-purpose hardware would accomplish the functions of one or two modules in the hierarchical control system architecture, and can thus fit very elegantly into a system that implements this architecture.

Another major problem of intelligent control is that of learning. Learning and the ability to generalize are very important for autonomous systems that must operate efficiently in a wide variety of situations in a complex real-world environment. We feel that neural network systems offer promising approaches to this problem. Again, these systems can fit very nicely into our hierarchical control system architecture.

Acknowledgments

The following people at the National Institute of Standards and Technology made significant contributions to the MAUV project: Hoosh Abrishamian, Mike Ali, Shu-jen Chang, Ken Goodwin, Hui-Min Huang, Maris Juberts, Steve Legowik, Peter Mansbach, John Michaloski, Don Orser, Dave Oskard, Rick Quintero, Marty Roche, Mark Rosol, Scott Swetz, Tsung-Ming Tsai, Barry Warsaw, Tom Wheatley.

The following people at the University of New Hampshire contributed significantly to this project: Richard Blidberg, Jim Jalbert, Steve Chappell, Mike Shevenell, Dennis Stamulis, Bob Welsh, Rod Haywood.

Significant contributions were also provided by George Pugh and Joe Krupp of Decision Science Applications, and Bob Finkelstein of Robotic Technology, Inc.

References

1. Albus, J. S. "A Control System Architecture for Intelligent Machine Systems." *IEEE Conf. on Systems, Man, and Cybernetics*, Arlington, VA, October 1987.

2. Albus, J.S. "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles." NIST Technical Note 1251, National Institute of Standards and Technology, Gaithersburg, MD, September 1988.

3. Blidberg, D.R. "Guidance Control Architecture for the EAVE Vehicle." *IEEE Journal Oceanic Engineering*, October 1986.

4. Kent, E.W., Shneier, M.O. and Lumia, R. "PIPE (Pipelined Image Processing Engine)." *J. Parallel and Distributed Computing*, 2, 50-78, 1985.

5. Nilsson, N.J. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

6. Orser, D.J. and Roche, M. "The Extraction of Topographic Features in Support of Autonomous Underwater Vehicle Navigation." *Proc. Fifth International Symposium on Unmanned Untethered Submersible Technology*, University of New Hampshire, Durham, NH, June 1987.

7. Oskard, D.N., Hong, T.-H. and Shaffer, C.A. "Spatial Mapping System for Autonomous Underwater Vehicles." Proc. SPIE Symposium on Optical and Optoelectronic Engineering, Cambridge, MA, November 1988.

8. Pugh, G.E. and Krupp, J.C. "A Value-Driven Control System for the Coordination of Autonomous Cooperating Underwater Vehicles." *Proc. Fourteenth Annual Symposium of the Association for Unmanned Vehicle Systems*, Washington, D.C., July 1987.

9. Samet, H. "The Quadtree and Related Hierarchical Data Structures." *ACM Computing Surveys*, June 1984, 187-260.

10. Simpson, J.A., Hocken, R.J., and Albus, J.S. "The Automated Manufacturing Research Facility of the National Bureau of Standards." *Journal of Manufacturing Systems*, Vol. 1, No. 1, 1983.