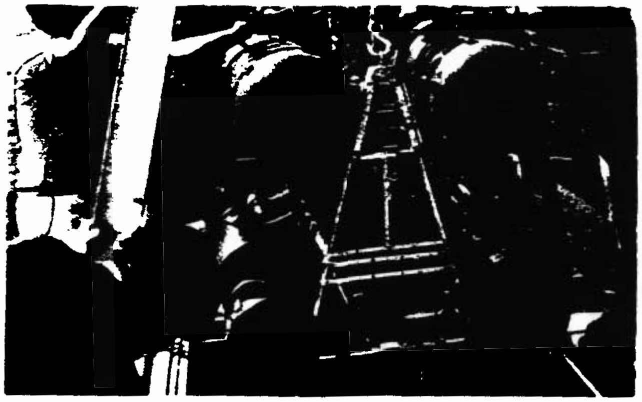
Unmanned Systems, Vol. 7, No. 1, Winter 1988-89.



Overview of MAUV: Multiple Autonomous Undersea Vehicles

by Martin Herman and James S. Albus

1. Introduction

The National Institute of Standards and Technology (NIST) Multiple Autonomous Undersea Vehicles (MAUV) project involved the development of a real-time intelligent control system that performs sensing, world modeling, planning and execution for undersea vehicles. The project was funded by the DARPA Naval Technology Office. The goal of the project was to have multiple vehicles exhibiting intelligent, autonomous, cooperative behavior. All software for controlling the vehicles reside on computer boards mounted on-board the vehicles.

This paper presents an overview of the project. It focuses on the hieararchical control system for controlling the vehicles, and describes how planning, execution and world modeling are done in the system. Further details on the project may be found in [2].

1.1 The MAUV Vehicles

Figure 1a and Figure 1b show a diagram and 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 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

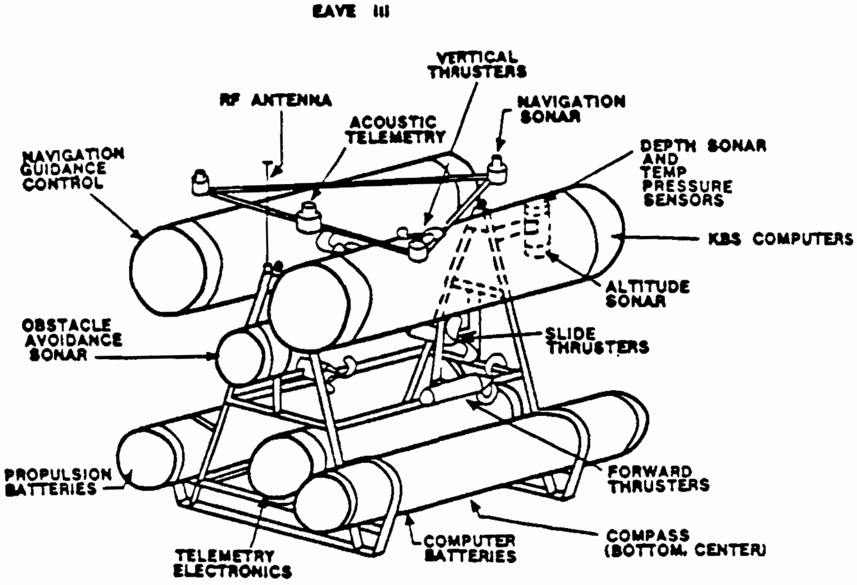


FIGURE 1a. Diagram of University of New Hampshire EAVE-EAST MAUV vehicle.

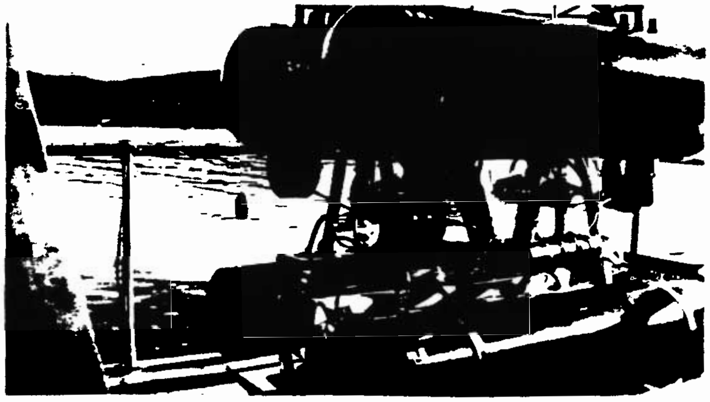


FIGURE 1b. Photograph showing University of New Hampshire EAVE-EAST MAUV vehicle.

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 are 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. The vehicles may be either near the water surface or near the lake bottom when performing the search.

The near-target maneuvers *scenario involves performing* triangulation maneuvers near a target either to localize it or to take pictures of it. Figure 2 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.

2. Hierarchical Control

The control system for the vehicles is hierarchically structured and is shown in Figure 3 [1,2]. This control system is based on the one developed for the Automated Manufacturing Research Facility at NIST [8]. It is divided into three main components, shown as columns in Figure 3. These are *sensory processing, world modeling,* and *task decomposition.* The goal of the task decomposition component is to perform real-time decomposition of task goals by means of real-time planning, execution and task monitoring. The world modeling component performs the following functions: (a) it maintains a central real-time database of information about the state of the world and the internal state of the system, (b) it updates this database with information from sensory processing, (c) it provides expectations of incoming sensory data, (d) it responds to queries from the task decomposition component based on information in the database and on evaluations of possible future states of the world. The goal of the sensory processing component is to detect and recognize patterns, events and objects, and to filter and integrate sensory information over space and time.

The world model serves as a buffer between the sensory processing component and the planners and executors of the task decomposition component. That is, queries about the world required for planning and execution are made to the world model, and sensory processing is used to update this world model.

The control system is divided hierarchically into several levels. We view this kind of hierarchical division as a means of converting broad, high-level goals into commands to actuators, motors, communication transducers, sonar transducers, etc.

(continued on next page)

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

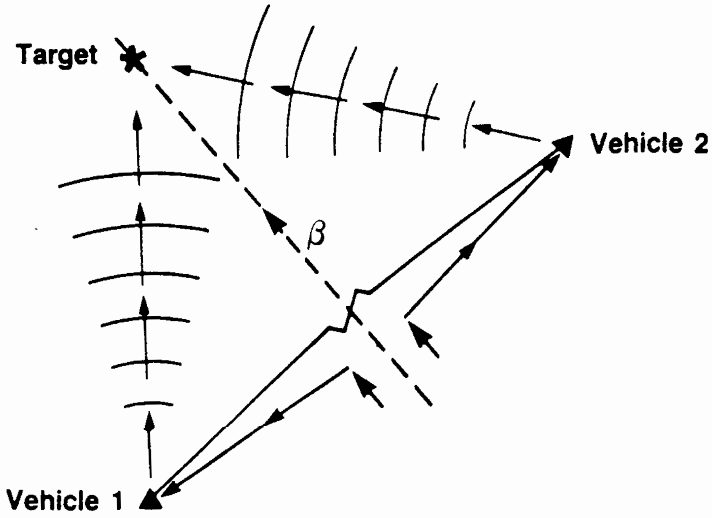


FIGURE 2. Target localization.

(continued from previous page)

In the task decomposition hierarchy, the highest level, the *mission* level, converts a command 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.

3. Hierarchical Planning and Execution

Before describing the elements of hiearchical planning and ex-

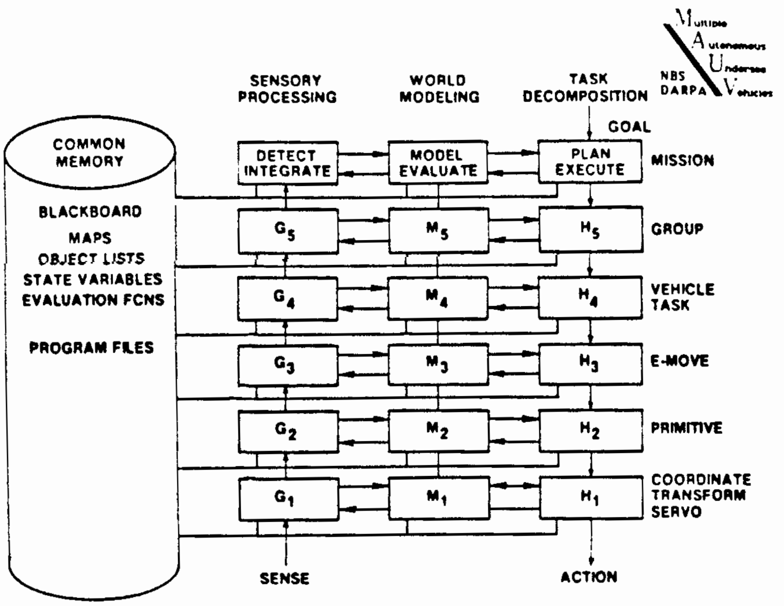


FIGURE 3. Block diagram of the NIST MAUV control system architecture.

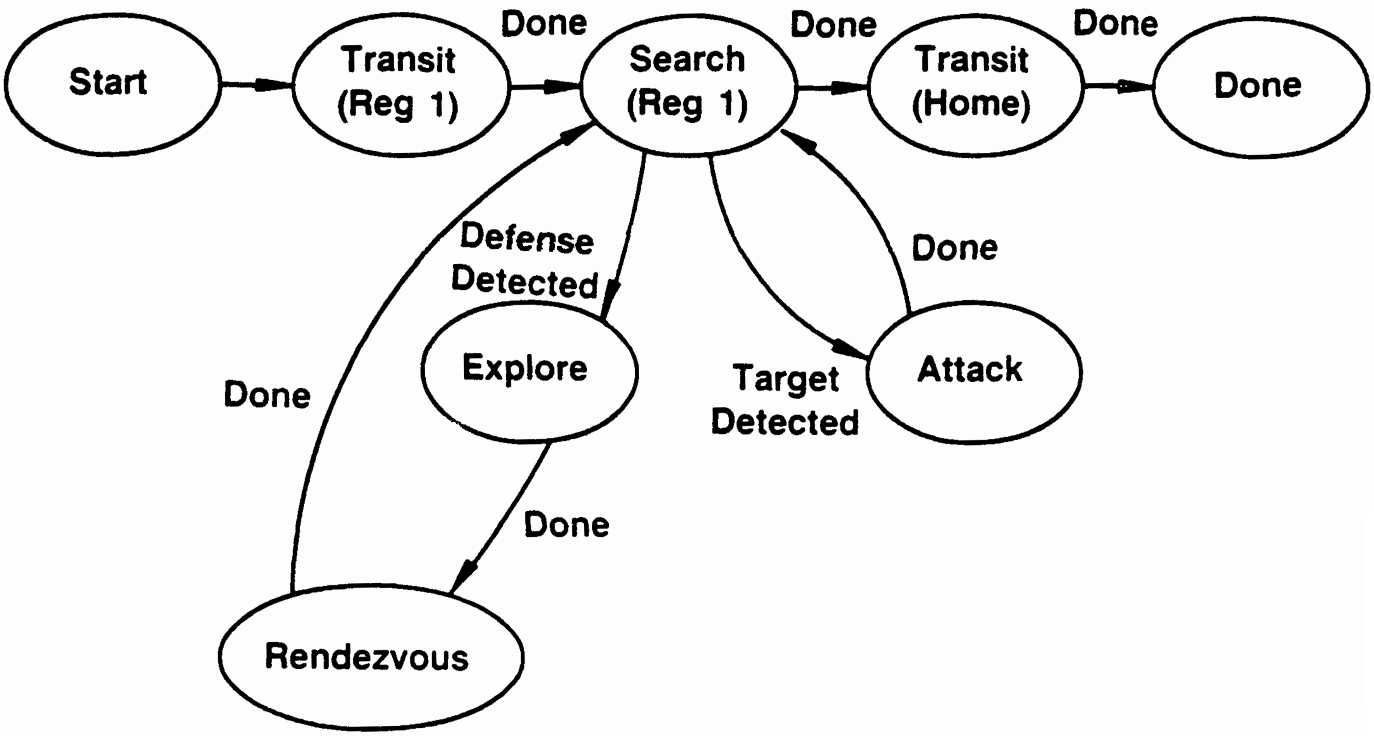


FIGURE 4. A plan graph for a mission level plan.

ecution, 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 4). 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.

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 5. 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 finely structured.

Figure 6 shows a single level of this hiearchy 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.

4. Levels in the MAUV Task Decomposition Hierarchy

The actual MAUV architecture is shown in Figure 7. Each large

(continued on next page)

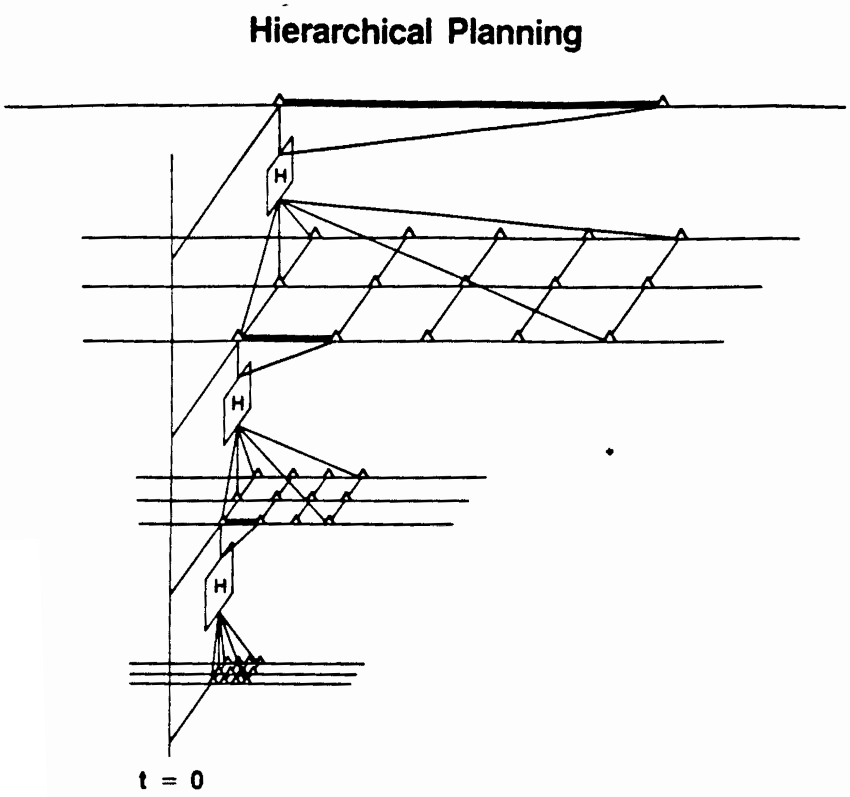


FIGURE 5. Three levels of real-time planning activity in the MAUV hierarchy.

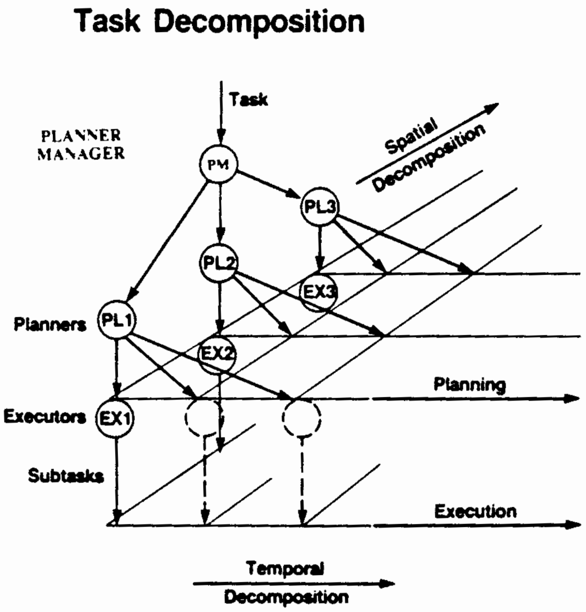


FIGURE 6. Internal structure of the task decomposition modules in the MAUV control system architecture at every level of the hierarchy.

(continued from previous page)

box in the figure 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.

4.1 Mission Level

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

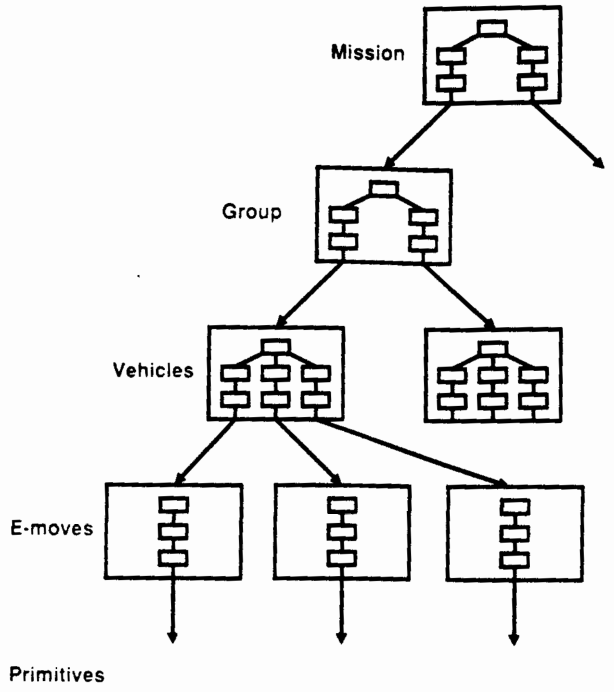


FIGURE 7. MAUV architecture.

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. Priorities 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 7, the mission level has a Planner Manager, a planner for each group, and an executor for each planner.

A flow chart for the mission level planner is shown in Figure 8. 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 course 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.

Replanning is done at regular intervals throughout the mission by repeating the program in Figure 8. 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 [7].

4.2 Group Level

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 will be

(continued on next page)

(continued from previous page)

used as weights in the cost function during A^* search[4].

The input group tasks define large scale actions to be performed by groups of MAUV vehicles. The function of the group level is to decompose these into sequences of tasks for individual vehicles. This level also attempts to maximize the effectiveness of the whole group by scheduling the actions of each vehicle so as to coordinate with the other vehicles in the group.

In our scenario, there is only one group of vehicles. As indicated in Figure 7, 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.

CONCEPTUAL FLOW CHART FOR MISSION LEVEL

(Group Task Planning)

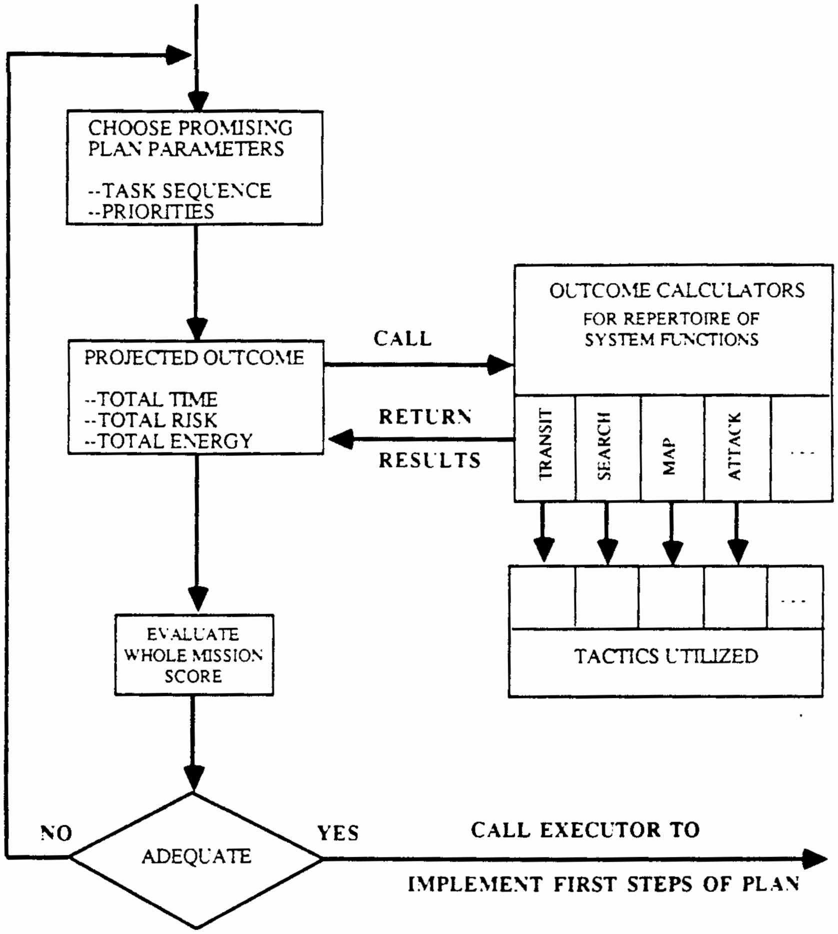


FIGURE 8. Mission level planner.

4.3 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 8, for each vehicle there is one Planner Manager, three planners (one for each subsystem), and three executors. 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. Currently, this schedule is extracted from a rule database. In the future, the schedule will be determined by computing the value of each message, its urgency, the risk of breaking communication 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 from 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.

4.4 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 subystem), 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 7, 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 we currently consider 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.

5. Cooperative Vehicle Behavior

Cooperative behavior between the two 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 mis-

(continued on next page)

(continued from previous page)

sion 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 sonobuoy 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 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 canot occur because of stealth requirements or because a vehicle is out of communication range,

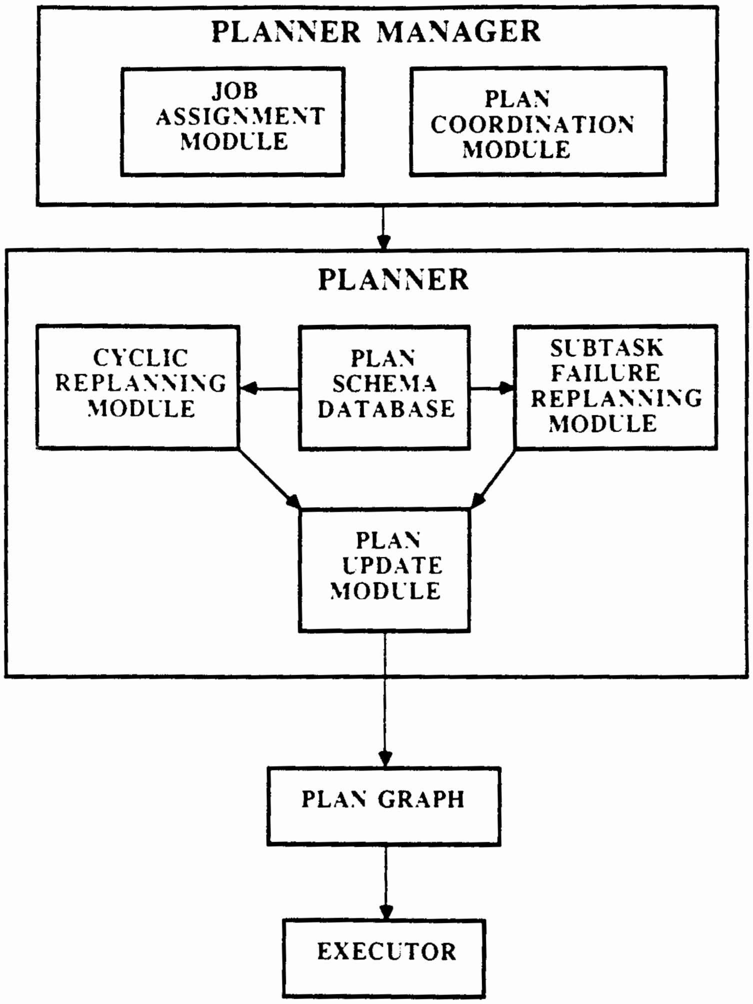


FIGURE 9. Group and vehicle level planner.

then each vehicle still has mission and group level software and can generate its own plans. Of course, this could lead to noncooperative maneuvers. Once communication is re-established, the mission and group leaders can take over.

6. 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 9, 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 9). 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 standard 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 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 occured. 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 latest time by 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 9), 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

(continued on next page)

(continued from previous page)

modified by the Subtask Failure Replanning Module (Figure 9). 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. Plan Schemas

A plan schemas 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 10, 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 10, for example, the GO-

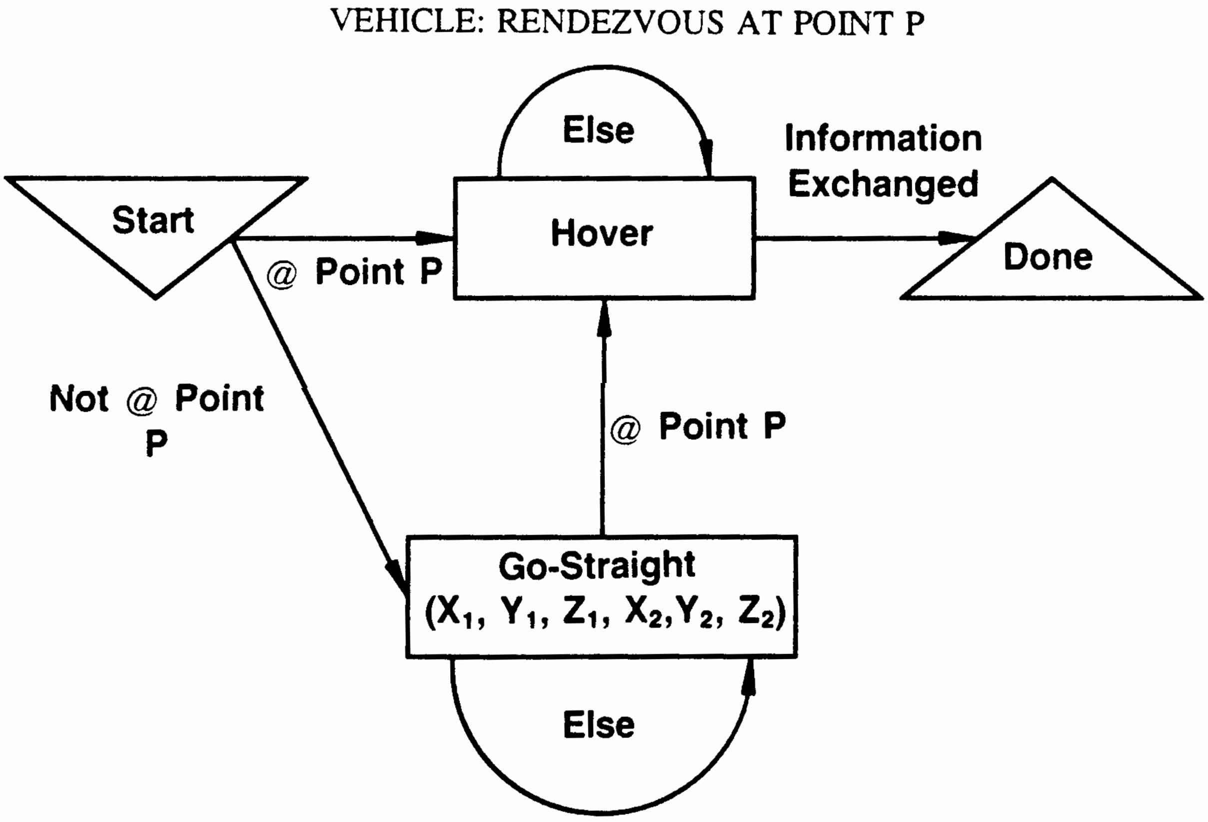


FIGURE 10. Vehicle level plan schema for "Rendezvous at point P."

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 applied to all future nodes of the plan schema that will be traversed by the interpreter, even though these nodes also have their own context subroutine components.

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 10, 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-STRAIGHT's, 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 scheme 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 11, 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.

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 consists of two main portions, (1) a set of databases containing knowledge about the state of the world and the internal state of the control system, and (2) procedures which update the databases, make predictions based on the databases, and search the databases to respond to queries from the planners and executors.

The set of databases in the world model may be divided into two portions, (1) a set of local databases, one for each level of the hierarchy and (2) a global database. Our current implementation has three local databases, one for the e-move level, one for the vehicle and group levels, and one for the mission level. (In principle, the local databases at the vehicle and group levels should be different; they are combined in the implementation for the sake of convenience in our particular scenario.) Each local database consists of a local map centered on the vehicle. As the vehicle moves, the local map moves with it, so that it is always centered on (or near) the vehicle. The local map is a digital terrain elevation map of the lake bottom—each square of a two-dimensional grid contains the elevation at that square. The set of local maps form a multi-resolution hiearchy. Resolution increases at each successively lower level, with the highest resolution at the e-move level. On the other hand, the region covered by the map increases at each successively higher level, with the mission level map covering the whole mission area.

The local map at the e-move level covers a 128 × 128 meter area. Each grid square represents a 0.5 × 0.5 meter area. Associated with each square is the estimated elevation along with a confidence for that elevation. Information in the e-move map is initially obtained from the global database, and is updated with new information from the obstacle avoidance and altitude sonars.

The local map at the vehicle and group levels covers a 512 × 512 meter area. Each grid square represents a 4 × 4 meter area. This map overlaps the e-move local map in such a way that each square in this map corresponds to a unique 8 × 8 grid square area in the e-move local map. Associated with each square of the vehicle and group local map is the average elevation, the standard deviation of this value, and the maximum and minimum elevations. These values are initially obtained from the global database, but their updates are computed from the e-move level map as the latter is updated.

The local map at the mission level covers a 1632 × 1632 meter area. Each grid square represents

(continued on next page)

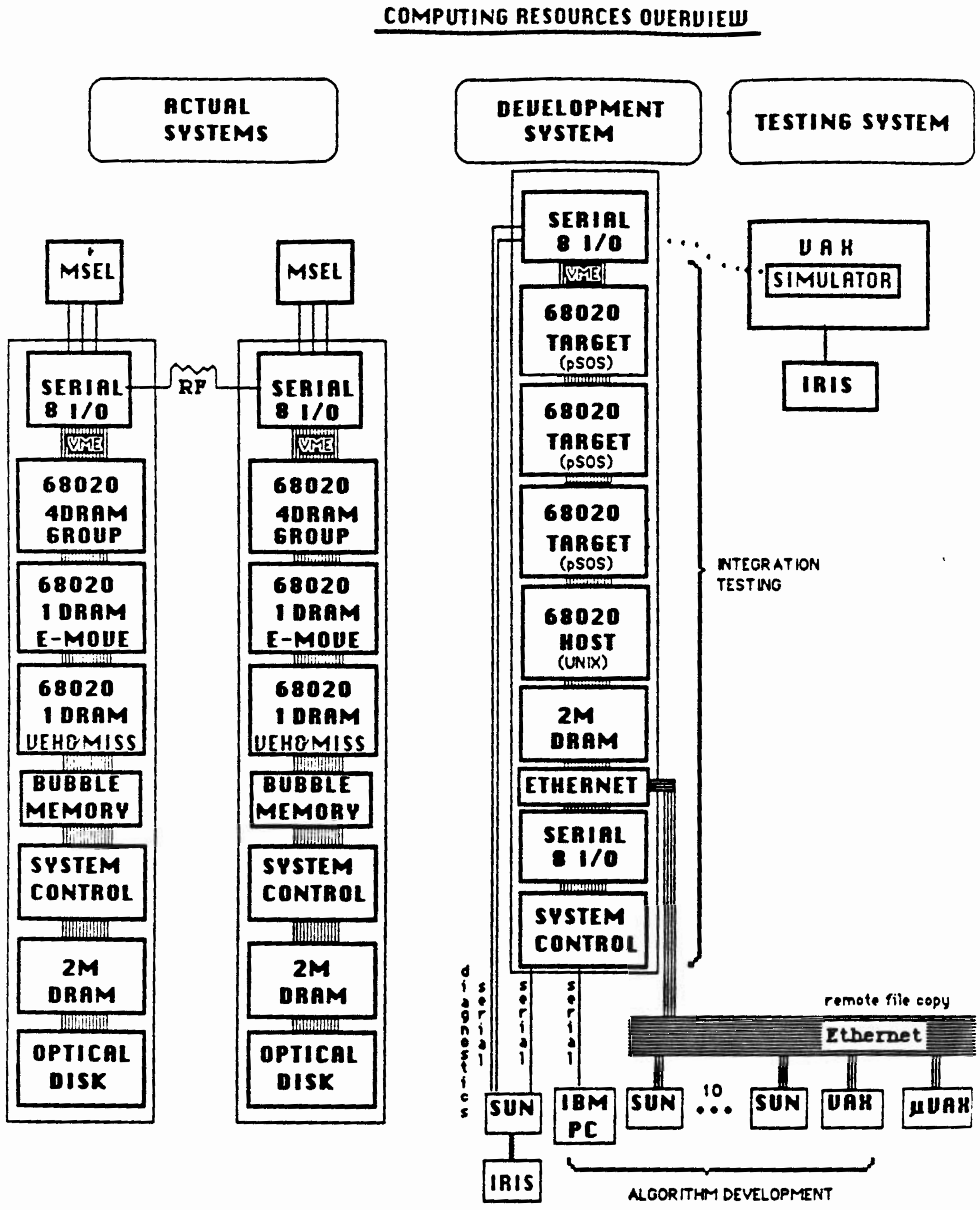


FIGURE 11. On the left is the target hardware for the two MAUV vehicles. On the right is the MAUV software development and simulation environment.

(continued from previous page)

an 8 × 8 meter area, and contains the average elevation. The values in this map are initially obtained from the global database and their updates are computed from the vehicle and group level map as the latter is updated.

The global database contains three elements, (1) an *a priori* map of the area in which the mission is performed, (2) a dynamically changing global map of newly obtained information, called a *sensory map*, and (3) a set of the significant objects in the world. The *a priori* map is a low resolution map of the lake bottom terrain, represented as a quadtree, which provides elevation data about the mission area. The sensory map, also represented as a quadtree, has a resolution equal to that of the e-move map. The sensory map is updated with new information from the e-move map.

The interactions between the global and local databases are as follows. As the vehicle moves through the mission area, the local maps must be shifted from time to time so that these maps remain centered on or near the vehicle. The frequency of shifting increases with successively lower levels because the apparent speed of vehicle motion across the map becomes greater with successively lower levels. Shifting occurs as follows. First, if the map being shifted is the e-move map, all modifications to the map as a result of sonar data are transferred to the sensory map. Otherwise, no change is made to the sensory map. Next, the center of the local map is changed to the current position of the vehicle. This results in a new local map which is formed by initializing its values from the *a priori* map and the sensory map. Since the sensory map contains high resolution information only for regions actually sensed by the vehicle sonars, it will usually be very sparse.

The global database contains a set of significant objects in the world, including vehicles, defenses, targets, sonobuoy fields and minefields. Associated with each object are its properties such as position, orientation, velocity and size. These are modified whenever new information about them is obtained through the sensors. Further details about the world model may be found in [5,6].

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. The following shows these values for each level of the hierarchy:

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 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 planer therefore generates a plan that represents a decom-

(continued on next page)

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

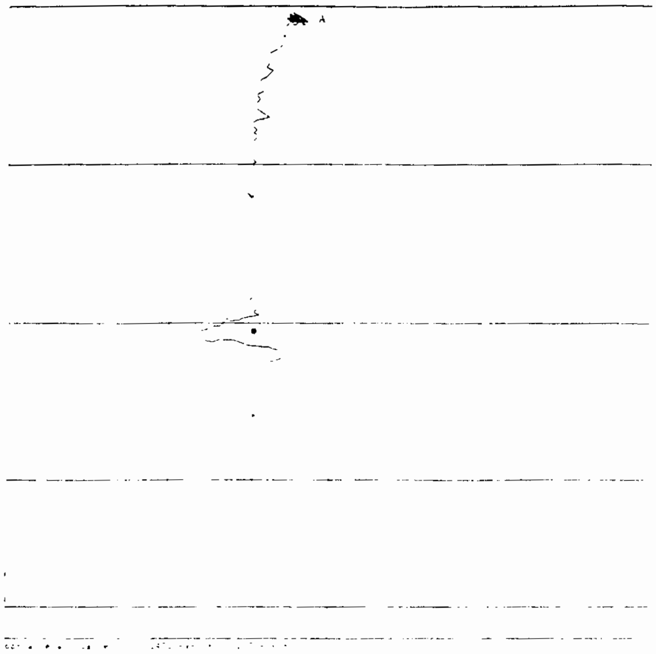


FIGURE 12. Obstacle avoidance test run.

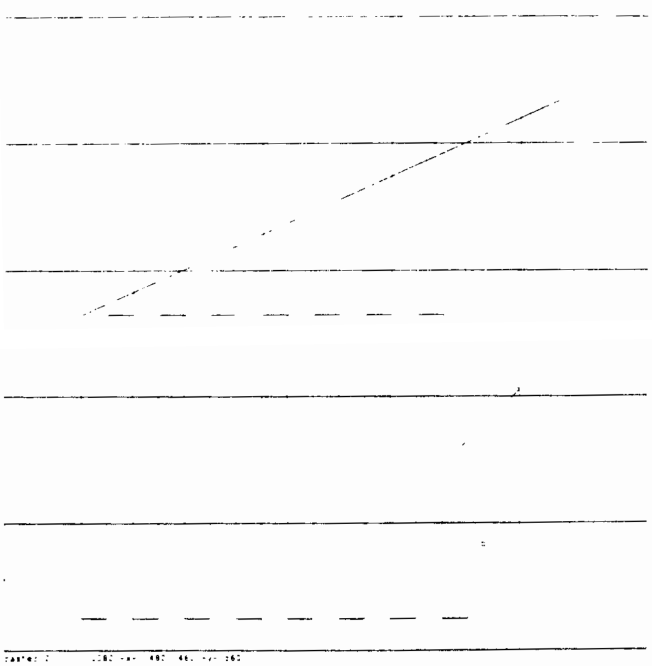


FIGURE 13. Predefined raster scan path.

(continued from previous page)

position 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 11. 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 11 is the software development and simulation environment. A variety of computers, including Sun workstations, a VAX 11/785, a microVAX, 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 down-loaded into the 68020 target hardware for real-time execution.

11. Experimental Results

This section describes some initial 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 scenarious 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 12 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 successfully planned and executed a path around the obstacle at point C.

The second experiment involved following along a predefined path. Figure 13 shows a raster-scan path from point A to point B. The vehicle determines 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 14. 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 15 shows three graphs. The top and middle graphs display the x and y positions, respectively, of the vehicle 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.

Acknowledgements

The following people at the National Institute of Standards and Technology made significant con-

(continued on next page)

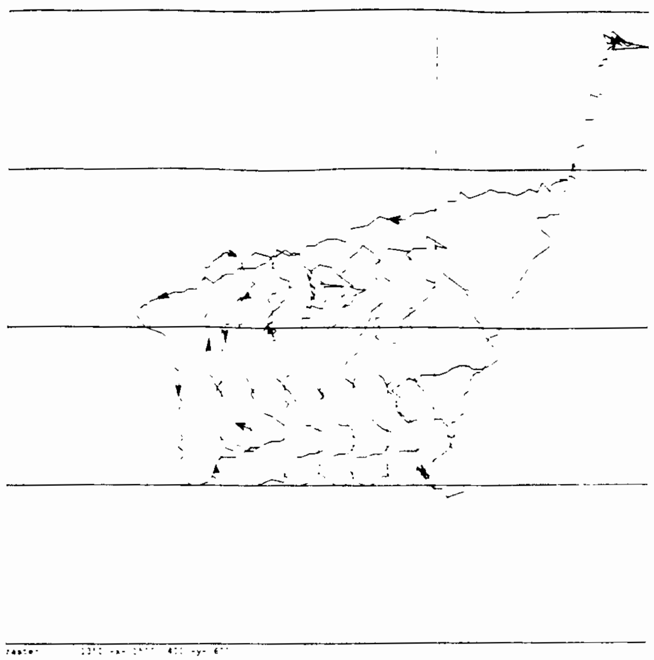


FIGURE 14. **Actual raster scan path executed. Arrows show directions travelled.**

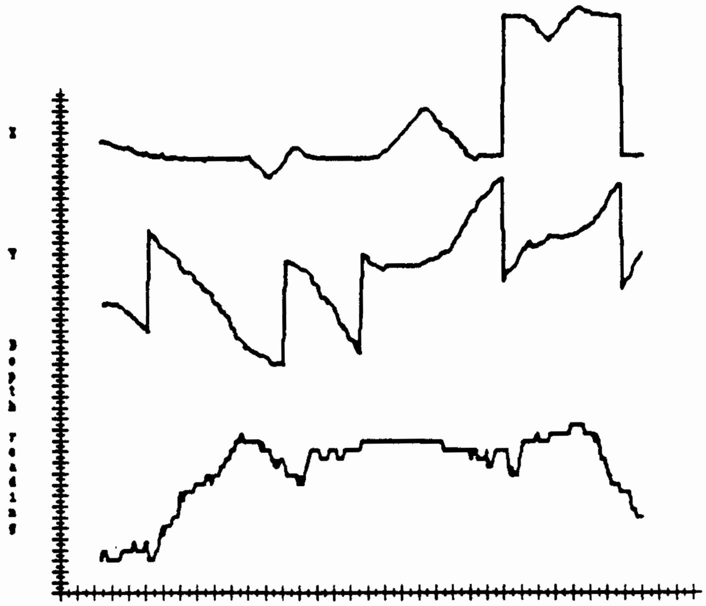


FIGURE 15. **Updating the world model lake depth.**

(continued from previous page)

tributions to the MAUV project: Hoosh Abrishamian, Mike Ali, Shu-jen Chang, Ken Goodwin, Tsai-Hong Hong, Hui-Min Huang, Maris Juberts, Steve Legowik, Peter Mansbach, John Michaloski, Don Orser, Dave Oskard, Rick Quintero, Marty Roche, Mark Rosol, Scot Swetz, Tsung-Ming Tsai, Barry Warsaw, Tom Wheatley. ■

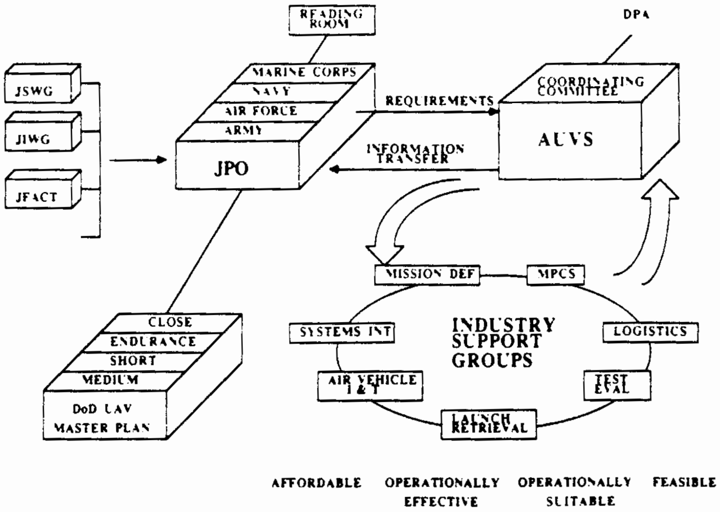
REFERENCES

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. Nilsson, N.J. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.
5. 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.
6. 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.
7. 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.*
8. 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.

AUVS Positioned as Interface Between Industry and JPO

At the 16 August 1988 AUVS/JPO Conference in Arlinton, Virginia, attendees were introduced to the newly formed Industry Support Group (ISG), established by AUVS to provide information in support of the DoD UAV Master Plan. The following flow chart defines AUV's interaction with the Unmanned Aerial Vehicle Joint Program Office and identifies various groups involved. Of particular interest is D.P. Associates, Inc., an Arlington-based firm which supports the JPO and acts as the interface for receiving and passing information to AUVS.

JPO/AUVS INTERACTION FOR UAV'S



Also in support of the JPO are military groups, including the Joint Services Working Group (JSWG), the Joint Interface Working Group (JIWG) and the Joint Field Activity Coordination Team (JFACT). Another item of interest is the JPO Reading Room, a repository for all JPO/ISG documents, currently scheduled for opening in late Spring. The Reading Room will be available to all AUVS and ISG members.

If you would like to know more about AUVS or the ISG Committees, call the AUVS national office at (202) 429-9440.