U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
(formerly National Bureau of Standards)

NIST Technical Note 1251

System Description and Design Architecture for Multiple Autonomous Undersea Vehicles

James S. Albus

Multiple
Autonomous
Undersea
Vehicles

NIST
DARPA

NIST Technical Note 1251

System Description and Design Architecture for Multiple Autonomous Undersea Vehicles

James S. Albus

Robot Systems Division
Center for Manufacturing Engineering
National Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

Sponsored by:

DARPA Defense Advanced Research Projects Agency
Office of Naval Technology
1400 Wilson Blvd.
Arlington, VA 22209-2308

September 1988

NOTE: As of 23 August 1988, the National Bureau of Standards (NBS) became the National Institute of Standards and Technology (NIST) when President Reagan signed into law the Omnibus Trade and Competitiveness Act.

U.S. Department of Commerce
C. William Verity, Secretary

National Institute of Standards and Technology
(formerly National Bureau of Standards)
Ernest Ambler, Director

National Institute of Standards and Technology
Technical Note 1251
Tech. Note 1251
126 pages (Sept. 1988)
CODEN: NBTNAE

U.S. Government Printing Office
Washington: 1988

For sale by the Superintendent of Documents,
U.S. Government Printing Office,
Washington, DC 20402

TABLE OF CONTENTS

Project MAUV
System Description and
Design Architecture

Executive Summary

The objective of the MAUV project was to demonstrate intelligent cooperative behavior in multiple autonomous undersea vehicles.

The approach was to build a control system architecture which fully integrates concepts of artificial intelligence and game theory with those of modern control theory. The control system was designed to permit a team of cooperating intelligent vehicles to compete against a team of cooperating intelligent opponents in a real-time dynamic environment.

Among the significant technologies pursued are:

* Real-time planning, using game theory and value driven logic;
* Dynamic world modeling, using multi-dimensional world maps and a real-time object oriented database;
* Sensory data fusion, using egosphere representations, real-time model matching, and stereo/motion integration;
* Multiplayer gaming.

The Real-time Control System (RCS-3) developed for the MAUV project has an open system architecture. Each module has clearly specified function and I/O interfaces. Data flow and timing are also specified. As a result, it is easy to integrate software from multiple sources, to upgrade modules, and add new sensors. RCS-3 is one of a family of open system architectures being developed at the National Bureau of Standards for automated factories, telerobot manipulators, and unmanned vehicles.

Progress in FY87:

* Two autonomous underwater vehicles were constructed and equipped with a five beam obstacle avoidance sonar, altitude and depth sonars, an acoustic navigation system, pressure and temperature sensors, a communications system, a hierarchical control system, and intelligent software.
* A RCS-3 control system architecture was designed and constructed consisting of six layers of task decomposition, world modeling, and sensory processing. Functionality was defined and code written at all six levels. Code at the lowest three levels was integrated and tested on the vehicles in Lake Winnipesaukee, and code at the highest level was run in simulation.

* A real-time computer system was designed and constructed consisting of five CPUs per vehicle. This system uses a commercial real-time operating system with multi-tasking and multi-processors. The hardware consists of 68020 computer boards, four megabytes of RAM, and 400 megabytes of mass storage using optical disk technology. The hardware and operating system are capable of running both C and Lisp simultaneously with real-time communications between the C and Lisp programs. A network of 15 SUN computers was procured and assembled into a program development environment running under UNIX. Two sets of computer hardware were constructed and integrated into two vehicles. A simulator was developed and installed to run either on a SUN, on a micro VAX, or on the vehicle hardware.

Plans proposed for FY88:

* Fully integrate intelligent software at all six hierarchical levels.

* Design and build a four player gaming environment.

* Demonstrate multiplayer gaming for search and map (mine countermeasures), and search and attack (antisubmarine warfare) scenarios.

* Demonstrate the system operating on two vehicles in Lake Winnipesaukee.

* Transition the RCS-3 control system to a MK-30 target vehicle (in cooperation with the Naval Underwater Systems Center).

* Install a TV camera on at least one vehicle and perform real-time 3-D visually guided maneuvers.

* Specify the RCS-3 system to a sufficient level of detail to make it suitable to become a commercial product.

Funding during FY87 was $2.3 million. A decision was made by DARPA, Office of Naval Technology, in December 1987, to terminate the MAUV project due to lack of funding in FY88, and to attempt to transfer MAUV technology to other DARPA projects.

The purpose of this document is to decribe what was accomplished, what was planned, and what could be achieved if the approach taken here is pursued in the future by other DARPA projects.

Project MAUV
System Description and Design Architecture

1. Introduction

1.1 Objective

The objective of the MAUV project was to demonstrate intelligent cooperative behavior in multiple autonomous undersea vehicles.

1.2 Approach

The approach was to build a control system architecture which fully integrates concepts of artificial intelligence and game theory with those of modern control theory.

1.3 Research Issues

The research issues addressed by the MAUV project are: hierarchical distributed control, knowledge based systems, real-time planning, world modeling, value-driven reasoning, intelligent sensing and communication, gaming, and cooperative problem solving by two intelligent vehicles in a natural and potentially hostile environment.

At its most basic level, the MAUV project represents basic research on the nature of intelligent behavior. The scientific goal was the understanding of intelligence as a mechanism for acquiring and defending assets. The demonstration scenarios were designed to study, and attempt to mimic, aggression, predation, exploration, stealth, deception, escape, communication, and cooperation.

On another level, the MAUV project represents developmental research on potential applications of multiple autonomous vehicles to military operations. The objective was to understand the basic issues of intelligent cooperation between two or more autonomous vehicles in a hostile environment.

Intelligent cooperation requires that group goals transcend individual goals. Each vehicle must weigh the value of its own survival against the success of the mission. Risk must be weighed against probable payoff, and cost/benefit analysis must be factored into behavioral decisions.

Intelligent cooperation also requires communication. In a natural environment, communication is not always possible or reliable. Bandwidth is usually limited, and in military operations, every transmission carries the risk of revealing information of more value to the enemy than to the sender or intended receiver.

Intelligent communication is a goal directed activity. Information is transmitted for a purpose. What information needs to be transmitted? when is it needed? and by whom? When is the value of a piece of information of sufficient value to incur the risk to survival of revealing one's presence by transmitting a message? What are communication strategies which balance risk against benefits?

There are also issues of command and control when communication is impossible or inadvisable. How should the control system be structured so that two or more vehicles can have equivalent intelligence when they are apart, but one vehicle is recognized as the leader of the pack when they are together? How do they share knowledge acquired by only one? What if they cannot agree on a strategy?

The initial focus of the MAUV project was on potential applications of two autonomous undersea vehicles. The types of scenarios that were being studied were:

a) One vehicle searches an area while the other relays messages about what has been found.
b) One vehicle illuminates a target while another takes action against the target.
c) One vehicle actively hunts for the target, while the other lies in wait.
d) One vehicle attracts the target's attention, while the other closes in for the kill.
e) One vehicle occupies the enemy defenses, while the other slips past unnoticed.
f) One vehicle draws attention to itself, while the other escapes with valuable information.

Future concepts were to be developed for more than two vehicles. These include tactics for hunting in packs, patrolling and guarding, and methods for saturation of enemy defenses. Studies were to be made of group tactics such as phase coordinated emission of acoustic energy to create phantom sources, and pseudorandom coordinated pulse transmission to confuse enemy attempts to track the source of acoustic emissions.

2. Demonstration Scenarios

The MAUV project planned to conduct a series of demonstrations by two autonomous underwater vehicles. The demonstration scenarios were grouped into two basic classes:

1) cooperative search and map scenarios
2) cooperative search and attack scenarios

The environment chosen for the MAUV demonstrations was Lake Winnipesaukee.

2.1 Search and Map (Mine Countermeasures)

The search and map scenarios were to mimic a harbor or coastal shallows survey mission. One of the principal applications of this class of scenarios is mine countermeasures. Figure 1 illustrates the concept of using two or more MAUVs to search and map shallow areas such as bays, gulfs, harbors, estuaries, and rivers. In this mission type, the MAUVs 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 inspect objects with particular characteristics.

The search and map scenarios were being designed to show MAUV capabilities to operate in either friendly or unfriendly waters. In friendly waters, the mission might be to sweep an area to assure that no enemy vehicles, mines, or listening devices are present. In enemy waters, the mission might be to map mine fields and find safe pathways through them without being detected. Enemy defenses were assumed to use both passive and active sonar. The vehicles were to sense and plot the position of "enemy positions" (simulated by acoustic beacons), and perform a number of maneuvers relative to known or suspected enemy positions.

In Lake Winnipesaukee demonstration scenarios, enemy targets and defenses were simulated by transponder buoys. Several stationary transponder buoys, and at least one moving buoy was to be used. A boat with a human operator was to be used for towing the moving transponder buoy.

Passive sonar was simulated by allowing the MAUV vehicles to use only bearing information from the enemy target and defense transponders. Active sonar was simulated by allowing the MAUV vehicles to use both range and bearing information from the enemy transponders.

In Lake Winnipesaukee, the modified EAVE-EAST vehicles were to execute a variety of search patterns, including several involving separation and rendezvous for exchange of information. As they move, the MAUVs were to compute maneuvering tactics which take into account bottom topology and simulated enemy positions.

The MAUV 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 MAUV vehicles planned to demonstrate tactics using bottom features for shadowing their movements from known or presumed enemy positions.

The selection of tactics were to be performed by four methods: a) rule based analysis of particular task

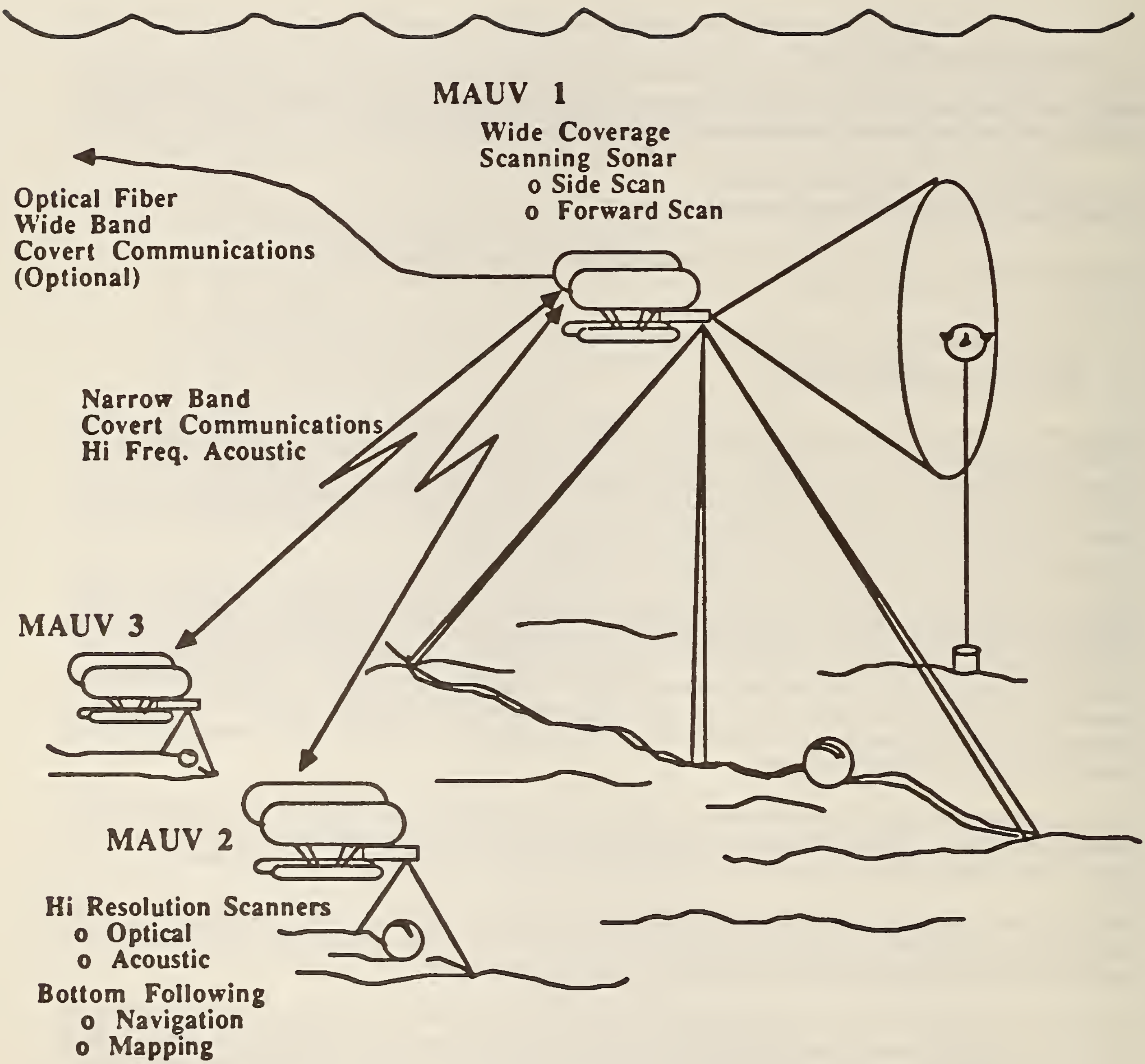


FIGURE 1. Illustration of MAUV search and map scenario.

situations, b) plan schemas, c) game theory algorithms, and d) AI search methods. The selection criteria was to be based on value driven logic which takes into account cost, risk, and payoff of various actions. This includes values placed on the vehicles, as well as the value of information and stealth. Value driven logic can generate strategies which vary from aggressive to conservative depending on priorities and values given to the mission.

2.2 Search and Attack (Antisubmarine warfare)

The second class of scenarios, search and attack, was designed to mimic deep ocean missions, and relatively little use was to be made of bottom features. The principal application of this class of scenarios was to be antisubmarine warfare. These scenarios were to demonstrate the concept of carrying sensors and weapons off-board from a manned submarine. In this type of mission, two or more MAUVs would act as sensor platforms, probes, or pathfinders for a simulated manned nuclear submarine.

A typical operational MAUV Search and Attack mission planned for two MAUV Probe vehicles, such as shown in Figure 2, was to carry sensors ahead, to the side, or behind a manned sub in order to search for, locate, and engage the enemy. Tactics were to be explored whereby the two vehicles conduct coordinated search, attack, decoy, escort, escape, and data relay maneuvers.

In practice, a manned sub would serve as command ship to the MAUVs. In this scenario type, two MAUV probes would patrol using a variety of tactics, such as leap-frog, fly-formation, split-circle-and-rendezvous, leader-follower, and high-low. The vehicles would execute cooperative search patterns such as criss-cross weave patterns ahead and to the sides of the manned sub. The MAUV probes would typically communicate only when they pass in close proximity to each other.

Upon detecting a target with passive sonar, two or more MAUV probes might split and encircle the target to better triangulate on its position. Tactics were to be studied where one vehicle illuminates the environment and the second vehicle observes the target. For example, one MAUV might actively ping the target, or paint it with light, while the other MAUV Probe would remain passive and compute the target position and trajectory. The second MAUV Probe might then communicate targeting information to the manned sub, or transmit target position and velocity to weapons launched by the manned sub.

MAUV vehicles could also perform simpler missions, such as monitoring acoustic emissions from the manned sub, or serving as communications messengers. They might also provide hook-up service to "telephone booths" moored to the ocean bottom.

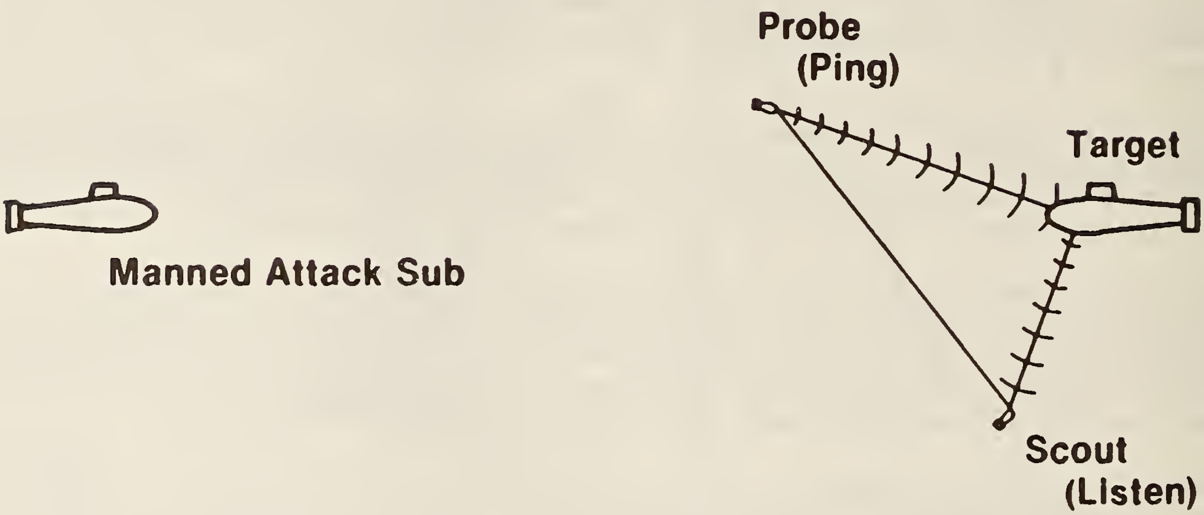
In a realistic scenario, MAUV probes would be ferried into action by a manned sub. They would be launched and recovered through torpedo tubes. The MAUVs would need to periodically return to the manned sub to be refueled.

One of the demonstrations to be conducted in Lake Winnipesaukee was to be rendezvous and docking. The two MAUV vehicles were to use sonar to rendezvous and optical tracking methods for docking. A series of tests was first to be performed in a test tank, and later in the lake. Both side-by-side and end-to-end docking were to be demonstrated.

2.3 Simulation/Gaming Environment

In addition to the demonstrations in Lake Winnipesaukee, a simulation/gaming system, such as shown

ANTISUBMARINE WARFARE



TARGET ACQUISITION/ATTACK SCENARIO

FIGURE 2: Target acquisition and attack scenario.

GAMING ENVIRONMENT

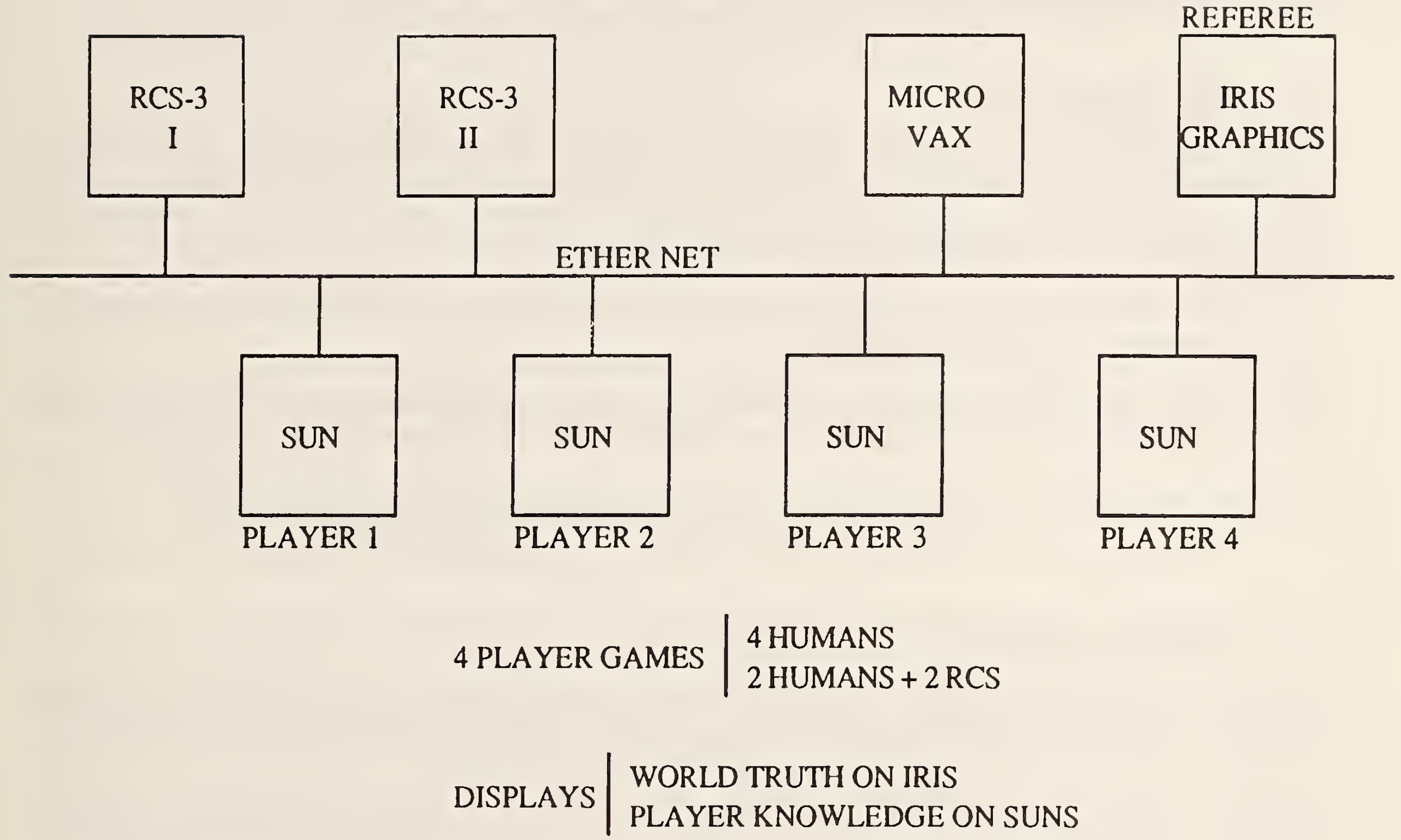


FIGURE 3: **A gaming environment for multiplayer games. Two Simulated MAUV vehicles under two RCS-3 controllers can play against two humans. For comparison two humans can play against two humans.**

in Figure 3, was planned for development. A complex archipelago of islands such as the passage between Northern Canada and Greenland was planned to be the environment.

2.3.1 The Gaming Goals

For the search and map scenario, the goal of the Blue MAUV vehicles would be to identify, photograph, and map the position of a stationary target object on the bottom of an area designated as a secret enclave. The Blue MAUV vehicles would carry no weapons, but have stealth properties which makes them difficult to observe except at very close range.

The goal of the Red manned vehicles was to destroy the Blue MAUV vehicles before they could gather their intelligence information and escape.

For the search and attack scenario, the goal of the Blue MAUV vehicles was to destroy a manned Red target vehicle being escorted through the archipelago by a pair of manned Red escort vehicles.

The goal of the Red manned vehicles would be to achieve safe passage for the target vehicle.

2.3.2 The Gaming Setup

The gaming was to take place in a simulated environment. The acoustic properties of the chosen region would be computed and stored in tabular form as a set of acoustic maps. About 400 points were to be defined in the region. For each point, a map could be defined as follows: Assume that an active ping is emitted. At each of the other points, plot the effect of that ping, using 8 bits to define the sound intensity, 8 bits to define the azimuth of the arriving wave front, and 8 bits to define the amount of reverberation. This will require 400 maps, each with 400 points, each with three bytes of data (480,000 bytes of memory).

There would exist a Truth database, to simulate the real world environment. Each of the four players, would have its own world model, which would be that player's best estimate of the state of the real world environment. Each human player would have both egosphere and world map displays. The egosphere would display raw or filtered sensor data overlaid on world model data.

The initial setup would use all human players, one human for each Red and each Blue vehicle. This would test the game for bugs, and establish a baseline for the expected outcomes of various strategies for both scenarios.

The final setup would have the Blue MAUV vehicles under control of the RCS control system, playing against humans controlling the Red vehicles.

The size of the geographical region was to be scaled, and the vehicles given fuel and speed capabilities such that the games could be conducted in a timely manner.

2.3.3 Capabilities and Attributes

a) For the search and map scenario:

The Red side would deploy 5 stationary passive sonar arrays. These would have coverage and sensitivity defined by the acoustics of their surroundings. These passive sonar arrays could measure bearing (with an accuracy defined by the acoustic surroundings) but not range. They might be confused as to azimuth by hearing more than one Blue vehicle at a time.

The Red side would also deploy active pingers with the passive arrays. These measure both range and bearing, with errors determined by the acoustic environment. They can recognize and identify multiple targets. These typically would be used only rarely, since they reveal the position of the sonar arrays.

The two Red vehicles would have passive sonars with 10x less sensitivity than the stationary arrays. They cannot measure range, and are less accurate than the stationary arrays in bearing measurements. They are more susceptible than the stationary arrays to confusion by more than one Blue vehicle.

The two Red vehicles would also have active scanning beams with controllable resolution. These can acquire acoustic range images with resolution of 1 degree per pixel out to 10 yards, 3 degrees out to 300 yards, and 10 degrees out to 1000 yards. Range and bearing errors would be determined by the acoustic environment.

The two Red vehicles would carry two simulated torpedoes with a range of 200 yards. The range and bearing of the targets must be precisely known before these weapons can be used effectively.

The two Red vehicles would have two way 300 baud communications with their home base via an RF antenna on a tethered float. Information from the stationary sensor arrays could be communicated via this link. The two Red vehicles would have two way 300 baud communication between each other via RF antenna. They would have two way 30-300 baud communications between each other via acoustic link provided there were line of sight and range less than 2 kilometers.

The two Blue vehicles would carry the same acoustic sensors and have the same communications capabilities as the Red vehicles.

The Blue vehicles would carry no simulated weapons, but would have visual and acoustic mapping sensors. They would have much lower observability than their Red pursuers. The Blue vehicles would also carry explosive charges that can be used to drown all Red sensors in reverberations for a period of minutes while the Blue vehicles make an attempt to escape or hide.

b) For the search and attack scenario:

The Blue rather than the Red side would possess the stationary sonar arrays.

The Red target vehicle would emit a characteristic sound the intensity of which would depend on the velocity of the target.

Both the Red and Blue vehicles would have the sensors described for the Red vehicles in the search and map scenarios. They would carry two simulated torpedoes with range of 220 yards. Both Red and Blue vehicles would have approximately the same observability under the same conditions.

3. Background

3.1 The MAUV Vehicles

The MAUV vehicles built for the Lake Winnipesaukee demonstrations are a second generation of the University of New Hampshire Marine Systems Engineering Laboratory EAVE-EAST design [1]. Figure 4a is a picture, and Figure 4b a diagram, of an EAVE-EAST MAUV vehicle. The vehicles were developed at the University of New Hampshire by Richard Blidberg and his associates. The vehicles are gravity stabilized in pitch and roll, with thrusters that allow them to be controlled in x, y, z, and yaw. They are battery powered with the batteries stored in cylindrical tanks at the bottom of each vehicle, and flotation tanks on the upper part of the vehicles. Each vehicle carries three acoustic navigation transponders which allow them to measure the range and bearing to navigation buoys placed in the water. Each vehicle also has a compass, pressure and temperature sensors, and bottom and surface sounders. In front, they have obstacle avoidance sonars 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. Each vehicle carries an radio frequency communications system and will soon also have an acoustic communications system.

3.2 The MAUV Control System

For the control system of the MAUV project, the National Bureau of Standards is designing and building RCS-3, a third generation of the NBS Real-time Control System (RCS) [2].

The first generation of RCS was a real-time sensory-interactive control system for a robot manipulator [3]. The second version (RCS-2) served as the control system architecture model for the NBS Automated Manufacturing Research Facility (AMRF) [4]. The current version (RCS-3) is being developed as the architecture for the MAUV project [5]. RCS-3 also forms the basis of the NASA/NBS Standard Reference Model Control System Architecture (NASREM) for the Space Station Flight Telerobot Servicer [6].

The RCS-3 control system architecture incorporates a number of concepts developed in previous and on-going robotics research programs, including the DARPA Autonomous Land Vehicle [7], the Air Force/DARPA Intelligent Task Automation program [8], the NASA telerobotics program [6], the supervisory control concepts pioneered by Sheridan at MIT [9], and the hierarchical control system developed for the NBS AMRF [10-12]. RCS-3 incorporates many artificial intelligence concepts such as goal decomposition, hierarchical real-time planning, model driven sensory processing, blackboards, and expert systems [13-19]. These are integrated into a systems framework with modern control concepts such as multivariant state space control, reference model adaptive control, dynamic optimization, and learning systems [20-23]. The RCS-3 architecture also readily accommodates concepts from operations research, differential games, utility theory, and value driven reasoning [24-25].

3.3 Institutional Participation

The Robot Systems Division of the National Bureau of Standards pursued the MAUV project because of its broad interest in performance measures and standards for intelligent machine systems. NBS is conducting research in advanced automation in several application areas: including manufacturing, construction, undersea vehicles, and space telerobotics. The MAUV project was of interest to NBS

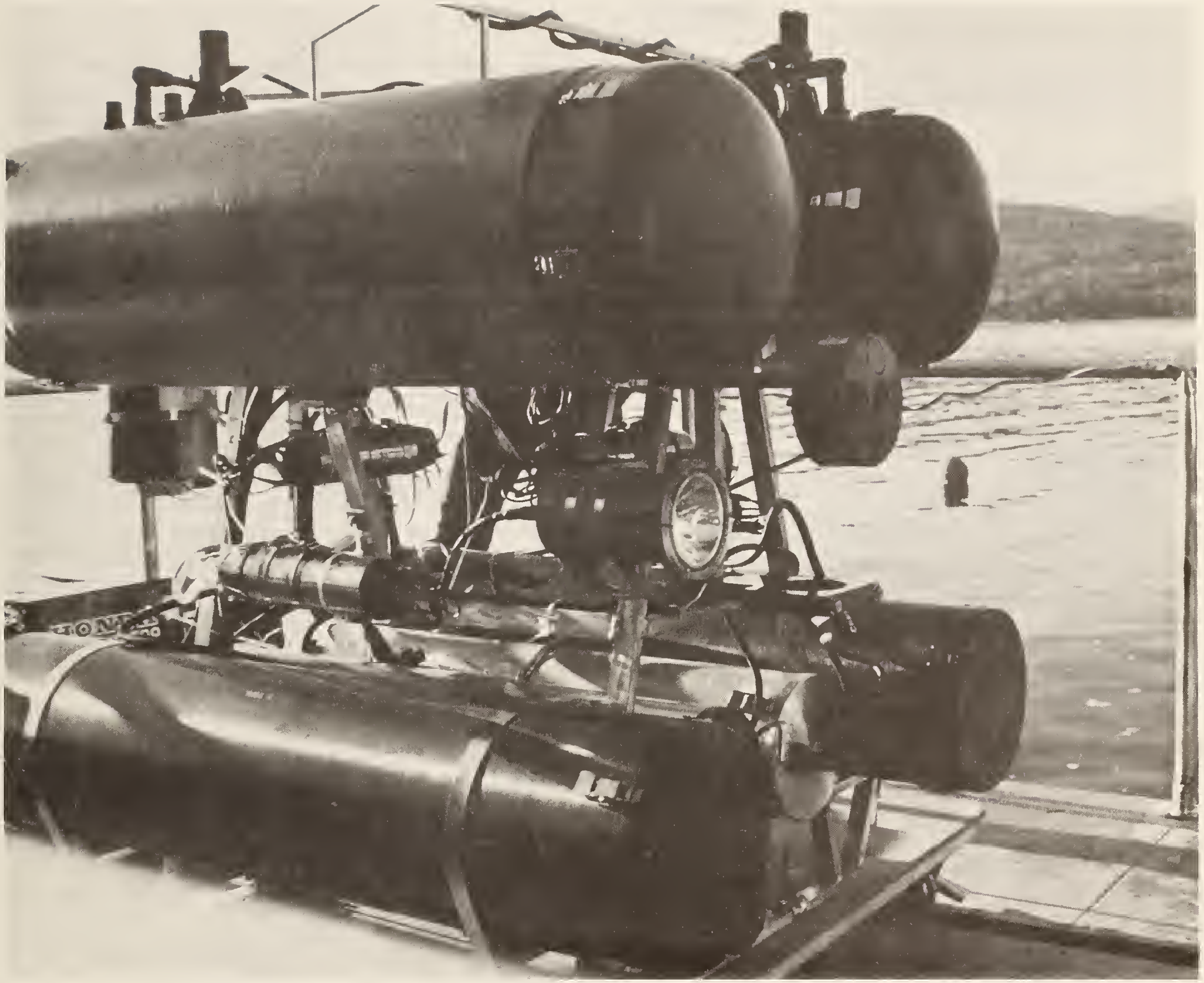


FIGURE 4A: University of New Hampshire Marine System Engineering Laboratory EAVE-EAST MAUV vehicle design.

EAVE III

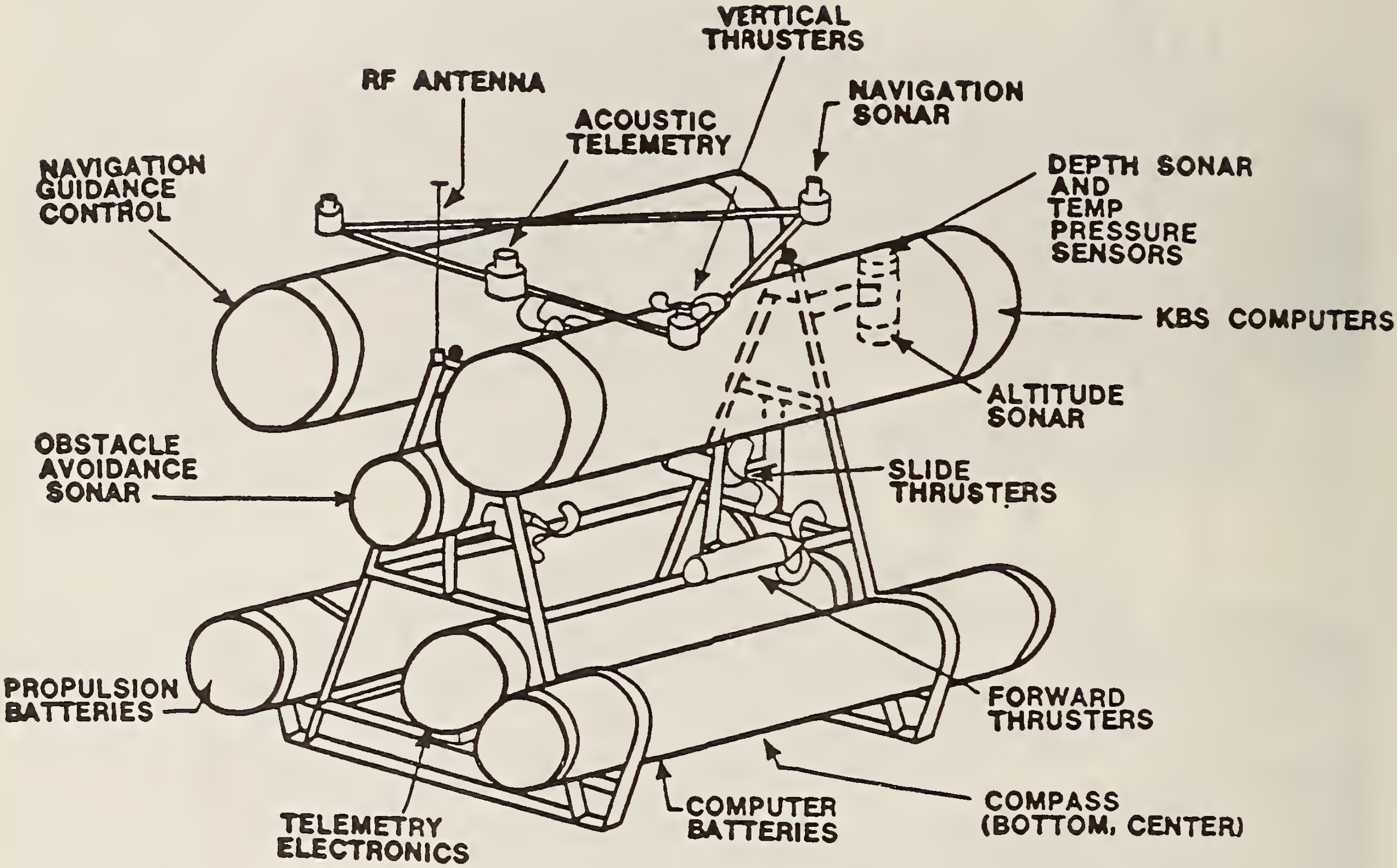


FIGURE 4B: Diagram of University of New Hampshire EAVE-EAST MAUV vehicle.

because autonomous undersea vehicles are members of the class of intelligent machines. The RCS-3 hierarchical control system architecture is being developed as a prototype for a proposed NBS Standard Reference Architecture Model for intelligent machine systems.

The University of New Hampshire Marine Systems Engineering Laboratory was involved because of its interest in autonomous undersea vehicles, and knowledge based systems for controlling them. UNH supplyed the vehicles, and the operational expertise in autonomous undersea vehicles. The Marine Systems Engineering Lab designed and built the EAVE-EAST MAUV vehicles, installed on them the lower two levels of the sensory processing and control system, and provided the interfaces to level three of the NBS RCS-3 system. UNH has also developed a high level Knowledge Based control System (KBS) which was demonstrated as a part of the MAUV project [26].

Also involved in the MAUV project was Professor Allen Waxman, of Boston University. He performed research on stereo vision for AUVs using the NBS Pipeline Image Processing Engine (PIPE) [27]. University of Maryland under Professor Azriel Rosenfeld conducted experiments on depth from image flow in the underwater environment, also using PIPE [28]. These capabilities were to be added to the MAUV vehicles in the later phases of the project. Lehigh University under Professors Roger Nagel and Glen Blank did studies of programming techniques for RCS-3 using state-graph techniques [29]. Decision Science Incorporated provided expertise in value driven logic for mission, group, and vehicle level planners [30]. Martin Marietta Baltimore provided an environmental simulator for scenario development [31]. Robot Technology Incorporated performed scenario development and developed performance evaluation techniques for MAUV demonstrations [32].

4. The MAUV Control System Architecture

4.1 The Control System Hierarchy

A high level block diagram of the MAUV RCS-3 control system architecture is shown in Figure 5. The system is a three legged, six level hierarchy of computing modules for task decomposition, world modeling, and sensory processing. This hierarchy is serviced by a communications system and a distributed common memory.

In the RCS-3 control system architecture the task decomposition modules perform real-time planning and task monitoring functions. Task goals are decomposed both spatially and temporally, as shown in Figure 6. The sensory processing modules filter, correlate, and integrate sensory information over both space and time so as to detect, recognize, and measure patterns, features, objects, events, and relationships in the external world. This is shown in Figure 7. The world modeling modules answer queries, make predictions, and compute evaluation functions on the state space defined by the information stored in global memory. The world modeling modules service both the task decomposition and sensory processing modules, as shown in Figure 8. Global memory is a database which contains the system's best estimate of the state of the external world. The world modeling modules keep the global memory database current and consistent.

4.2 Functional Levels in the MAUV Control Hierarchy

Figure 9 is a block diagram of the task decomposition hierarchy. 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 and actuators.

Figure 10 shows the relationship between the task decomposition, sensory processing, and world modeling modules.

At each of the six layers of the MAUV architecture a different function is performed.

Level 1 -- Coordinate Transform/Servo

Level 1 of the task decomposition hierarchy transforms coordinates from a vehicle coordinate frame into actuator coordinates. This level also servos thruster direction and actuator power. There is a level 1 planner and executor module for every motor and actuator in the MAUV vehicle. At this level in the sensory processing hierarchy, sensor readings are filtered, scaled, and entered into the world model as point readings. There is a level 1 module comparator and temporal integration for every sensor, including one for every pixel in the camera, or acoustic imaging system.

Level 2 -- Primitive (or Dynamic Level)

Level 2 works in vehicle or world coordinates. The task decomposition modules compute inertial dynamics, and generate smooth trajectory positions, velocities, accelerations for efficient vehicle maneuvers. In the sensory processing modules features of objects are recognized and stored in the world model as feature position, orientation, and velocity.

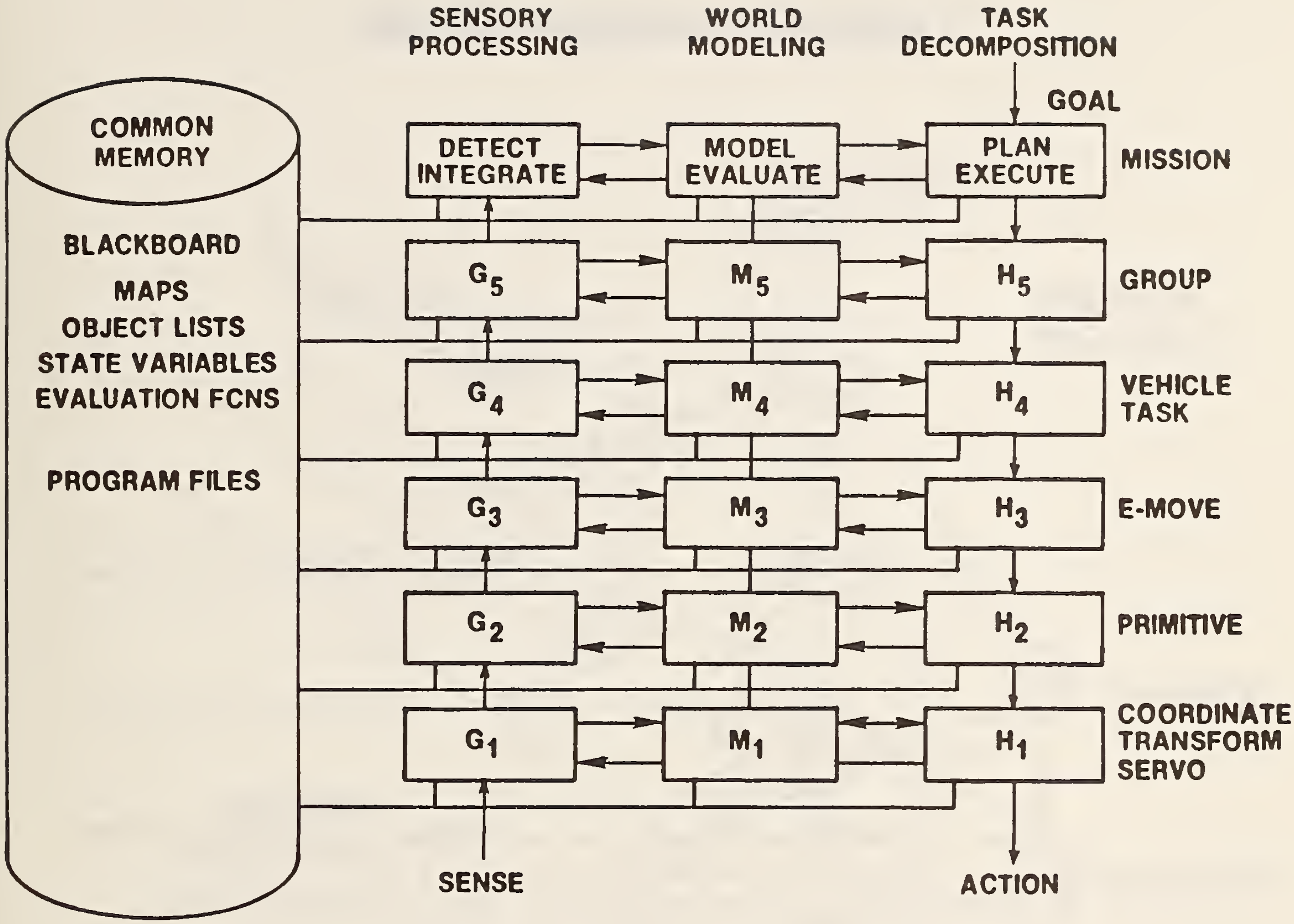


FIGURE 5: **High level block diagram of MAUV RCS-3 control system architecture.**

Task Decomposition

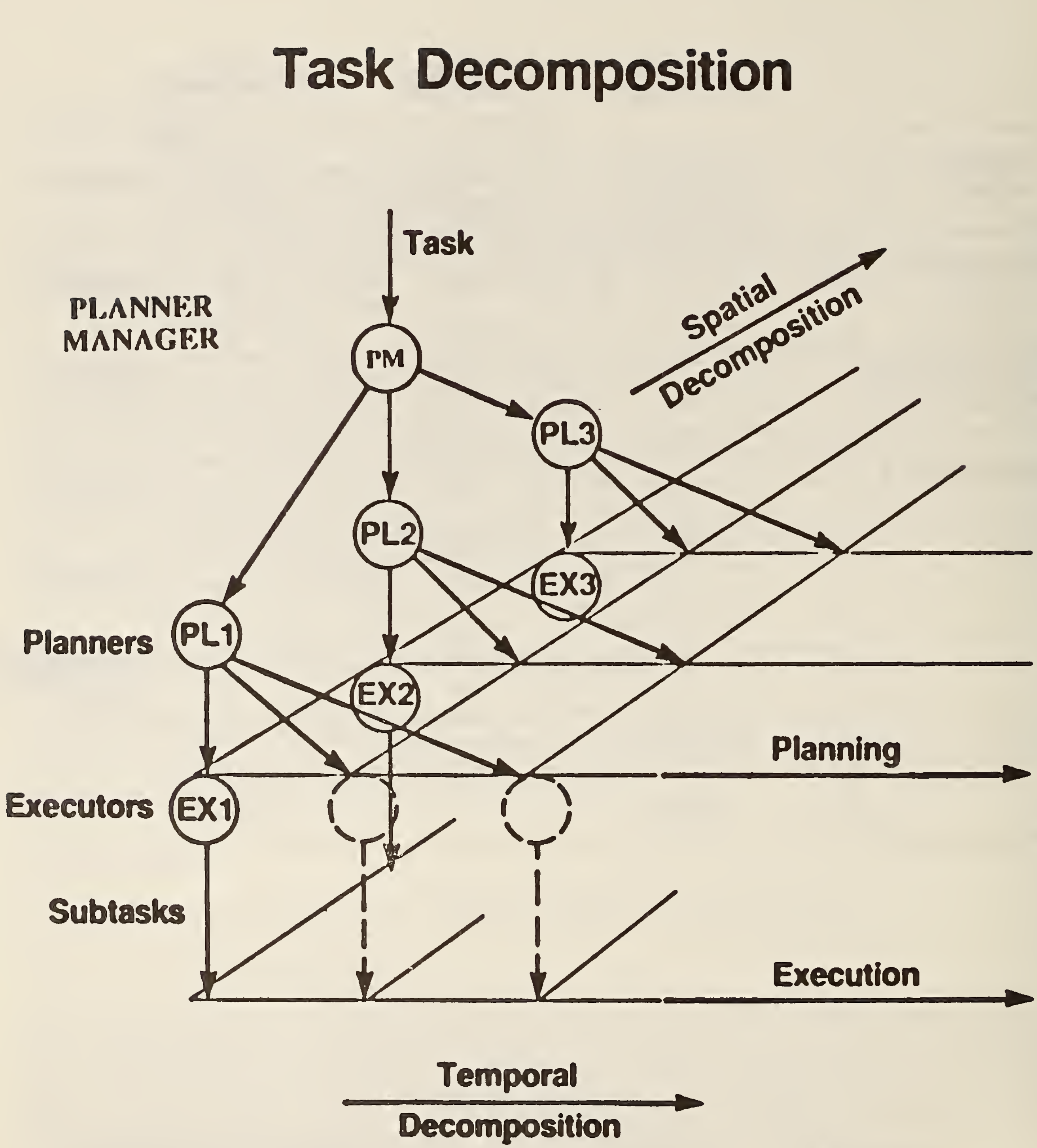


FIGURE 6: Each task decomposition H module decomposes task both spatially and temporally.

Sensory Processing

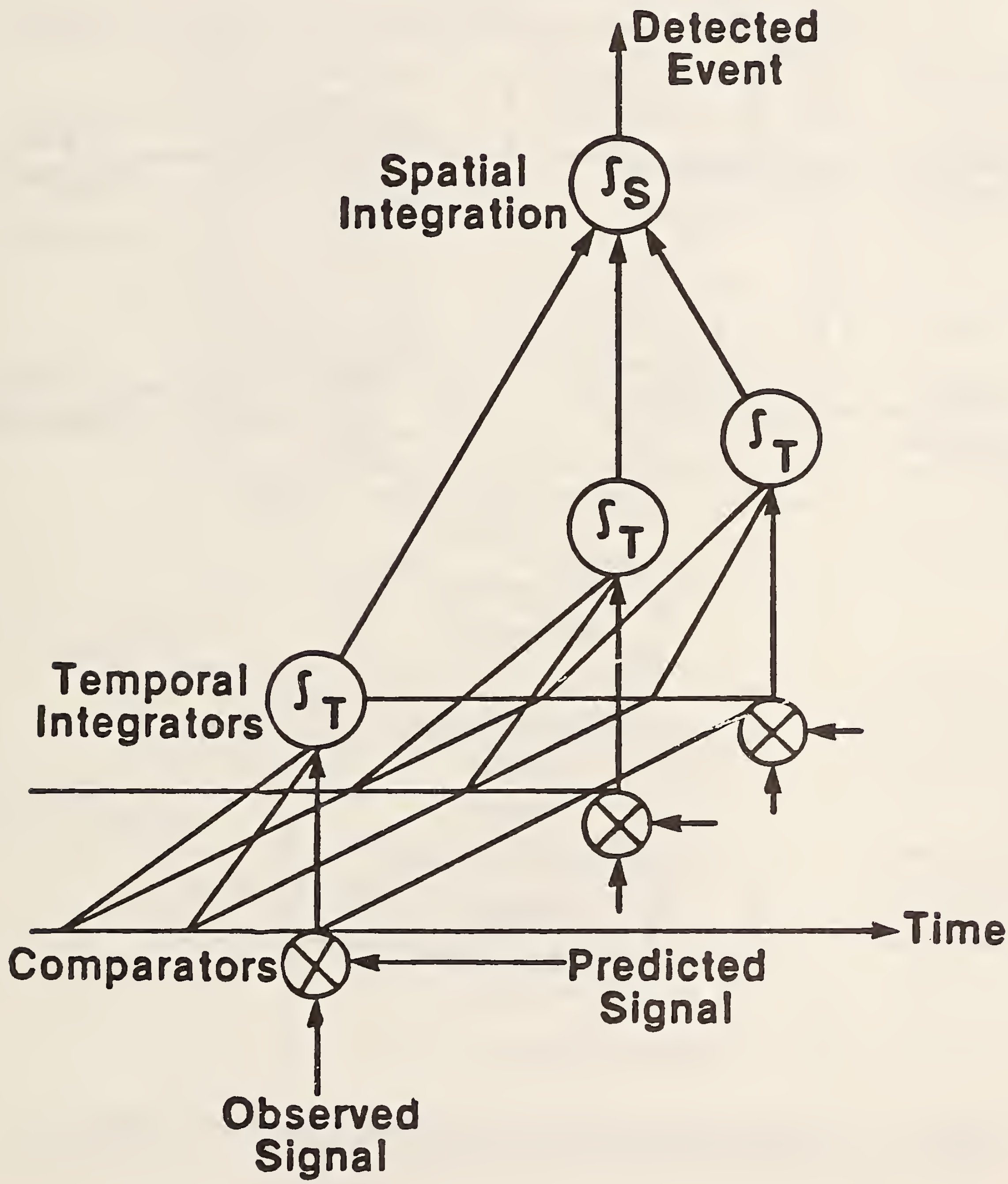


FIGURE 7: Each sensory processing G module compares observed signals with world model predictions and performs temporal and spatial integration.

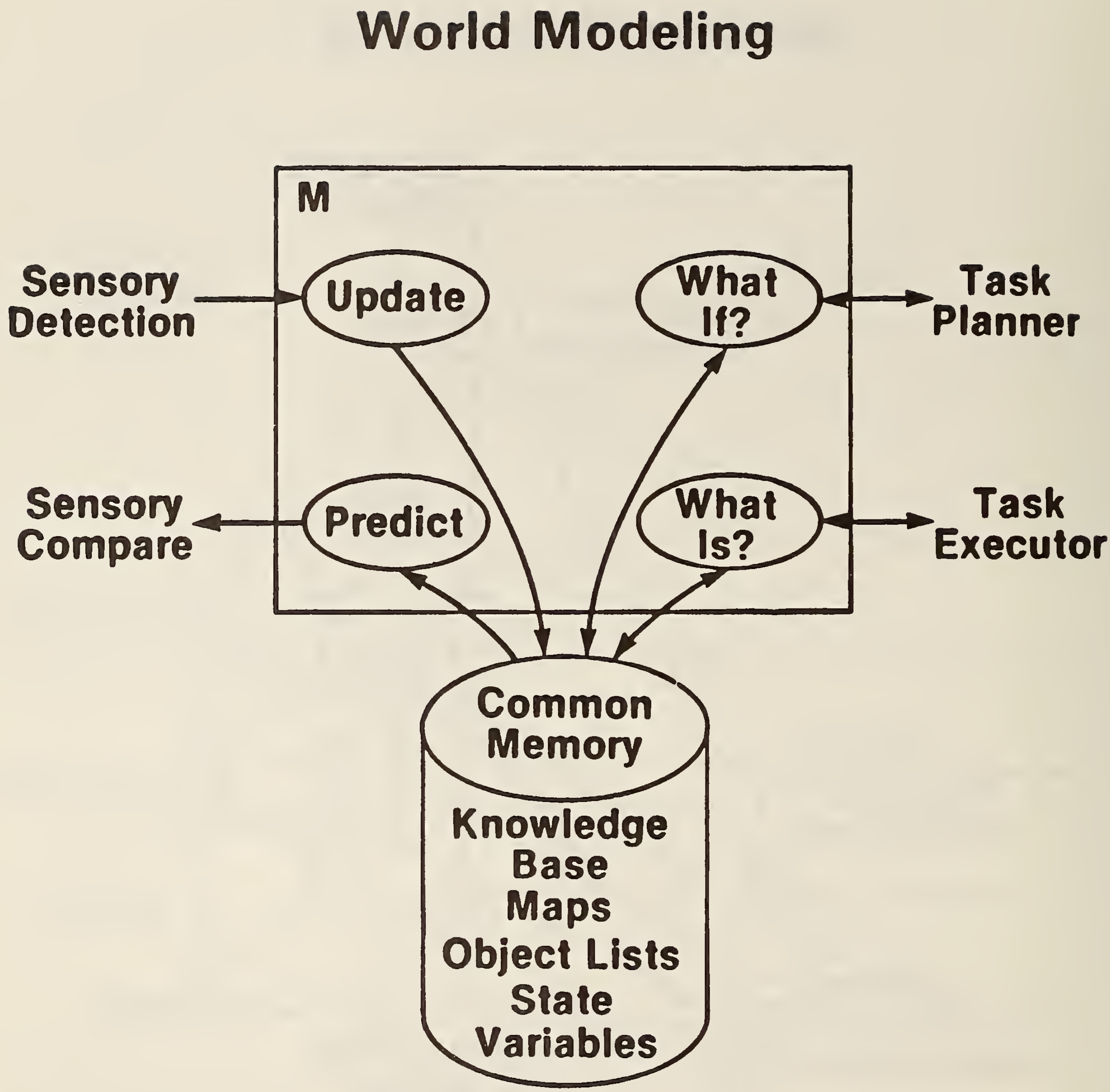


FIGURE 8: Functions performed by M modules in world model.

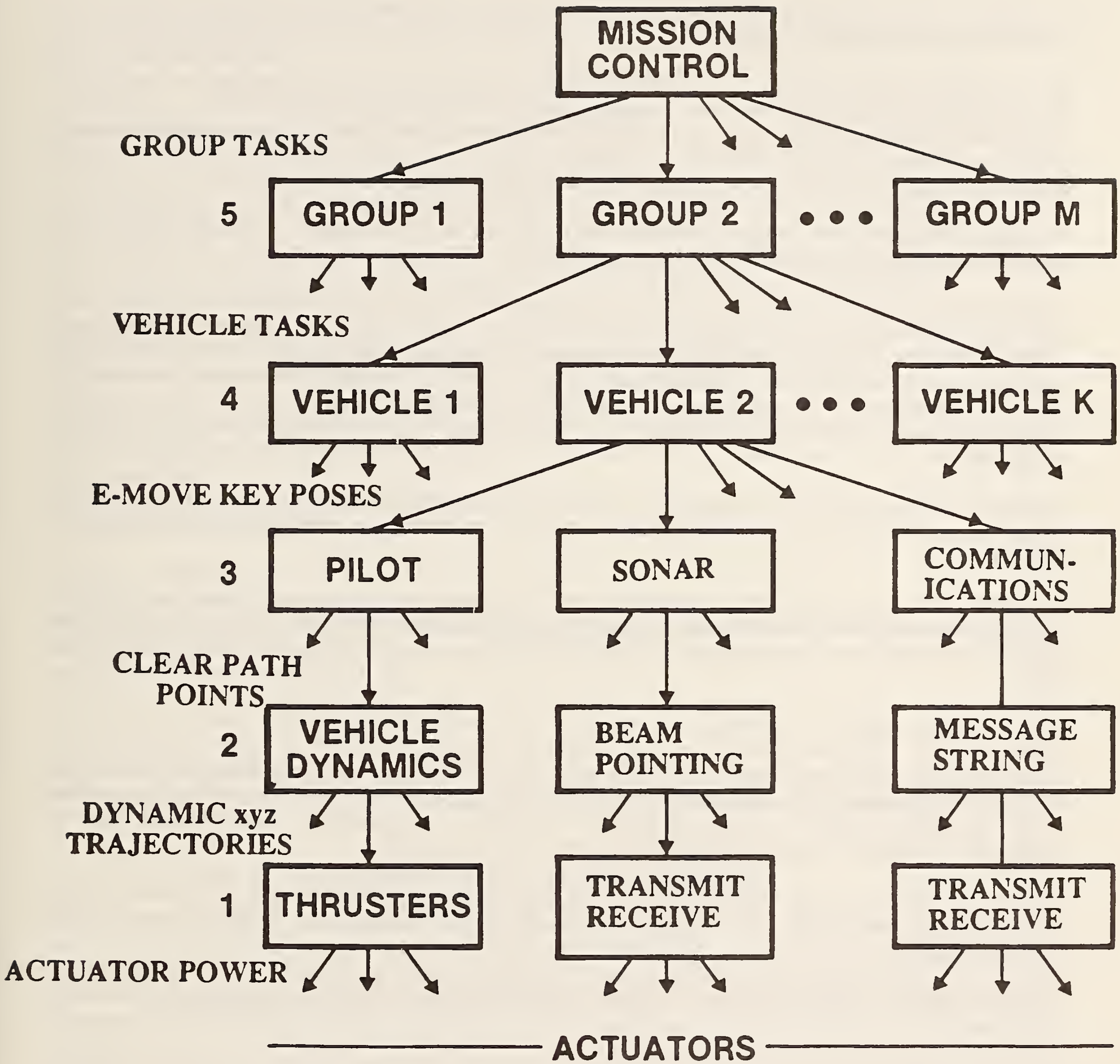


FIGURE 9: A block diagram of the MAUV task decomposition hierarchy.

Level 3 -- Elementary Move (E-move)

Level 3 works in both symbolic and geometric space. It decomposes elementary move commands (E-moves) into strings of intermediate poses, or primitive (dynamic) level commands. At level 3, objects are recognized and stored in the world model with position, orientation, and velocity. Coordinated movements of effectors are planned relative to surfaces of objects. Nearby obstacle surfaces are observed and avoided. Target object may be approached and touched.

As shown in Figure 9, each MAUV vehicle consists of three subsystems: pilot, communications, and sonar. E- moves are defined for each vehicle subsystem.

A pilot E-move can be defined as a smooth coordinated motion of the vehicle designed to achieve some position, orientation, or "key-frame pose" (see Section 11.4) in state-space, or space-time. The length of path defined as an E-move is typically the distance that can be directly observed by low resolution, wide angle, on-board sensors. The level 3 pilot planner computes clearance with obstacles sensed by on-board sensors and generates strings of intermediate poses that define motion pathways between key-frame poses. These intermediate poses become input commands to the level 2 dynamics computations.

A communications E-move is a message. The level 3 communications planner encodes messages into strings of symbols, adds redundancy for error detection and correction, and formats the symbols for transmission.

A sonar E-move may be defined as a temporal pattern of sonar pings or a scanning pattern for a passive listening beam designed to obtain a specific type of information about a feature of a specific target. The level 3 sonar planner decomposes sonar E-Moves into patterns of sonar pings and scanning beam dwell times.

Level 4 -- Vehicle Task

Level 4 works in object/task space. It decomposes vehicle commands, defined in terms of tasks to be performed by a single AUV on a single target object, into sequences of E-moves, defined in terms of vehicle subsystem actions on object features. At level 4, relationships between objects, and the properties of groups are recognized and entered into the world model.

The level 4 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 level 4 planners schedule sequences of E-Moves for the pilot, the communications, and the sonar subsystems.

The level 4 pilot planner uses world model maps to assure that there exists at least one pathway between keyframe poses. From map overlays, it estimates the cost, risk, and benefit of various routes and chooses a path that maximizes some cost-benefit evaluation function.

The level 4 communications planner schedules the messages to be sent. It computes the value of each message, its urgency, the risk of breaking communications silence, the power needed to make the message heard, and decides if and when to send the message.

The level 4 sonar planner analyzes the nature of the target, plans scanning patterns for passive or active beams, estimates the value of taking an active sonar sounding, and compares that against the risk of breaking silence.

Level 5 -- Group

Group task commands define actions to be performed cooperatively by groups of autonomous vehicles on multiple targets. The job assignment module of the level 5 planner manager decomposes group tasks into vehicle job assignments. 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.

Level 5 vehicle planners schedule group task lists into coordinated sequences of vehicle tasks. The vehicle planners use 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).

The level 5 plan coordinator constrains the actions of each MAUV to coordinate with the other MAUVs in the group so as to maximize the effectiveness of the MAUV group in accomplishing the group task goal.

They also estimate costs, risks, and benefits of various vehicle tactics (or task sequences).

Level 6 -- Mission

Missions are typically specified by a list of mission objectives, priorities, requirements, and time line constraints. Level 6 decomposes a commanded MAUV mission into a sequence of group tasks and assigns priorities and values to them.

The level 6 planning manager assigns vehicles to groups, sets priorities for group actions, and assigns mission objectives to MAUV groups. The level 6 planners schedule the activities of the groups so as to maximize the effectiveness of the mission. They also generate requirements for resources such as fuel, and time, develop a schedule, and set priorities for each respective group assignment.

4.3 Hierarchical versus Heterarchical (Horizontal) Organization

There has been considerable debate among experts in the field regarding the relative merits of hierarchical versus heterarchical control. Advocates of heterarchical control frequently characterize hierarchies as rigid and inflexible with overburdened central controllers and unintelligent peripheral elements. Advocates of hierarchical control often criticize heterarchies for requiring excessive amounts of communication and producing inefficient iterative solutions to temporal planning and resource allocation.

As shown in Figure 10, the organization of RCS-3 is both hierarchical and heterarchical (horizontal).

4.3.1 The Hierarchical Organization of RCS-3

RCS-3 is hierarchical in the sense that commands and status feedback flow hierarchically up and down the chain of command. It is also hierarchical in that sensory information is processed into increasingly higher levels of abstraction, and that information stored in the world model is organized hierarchically.

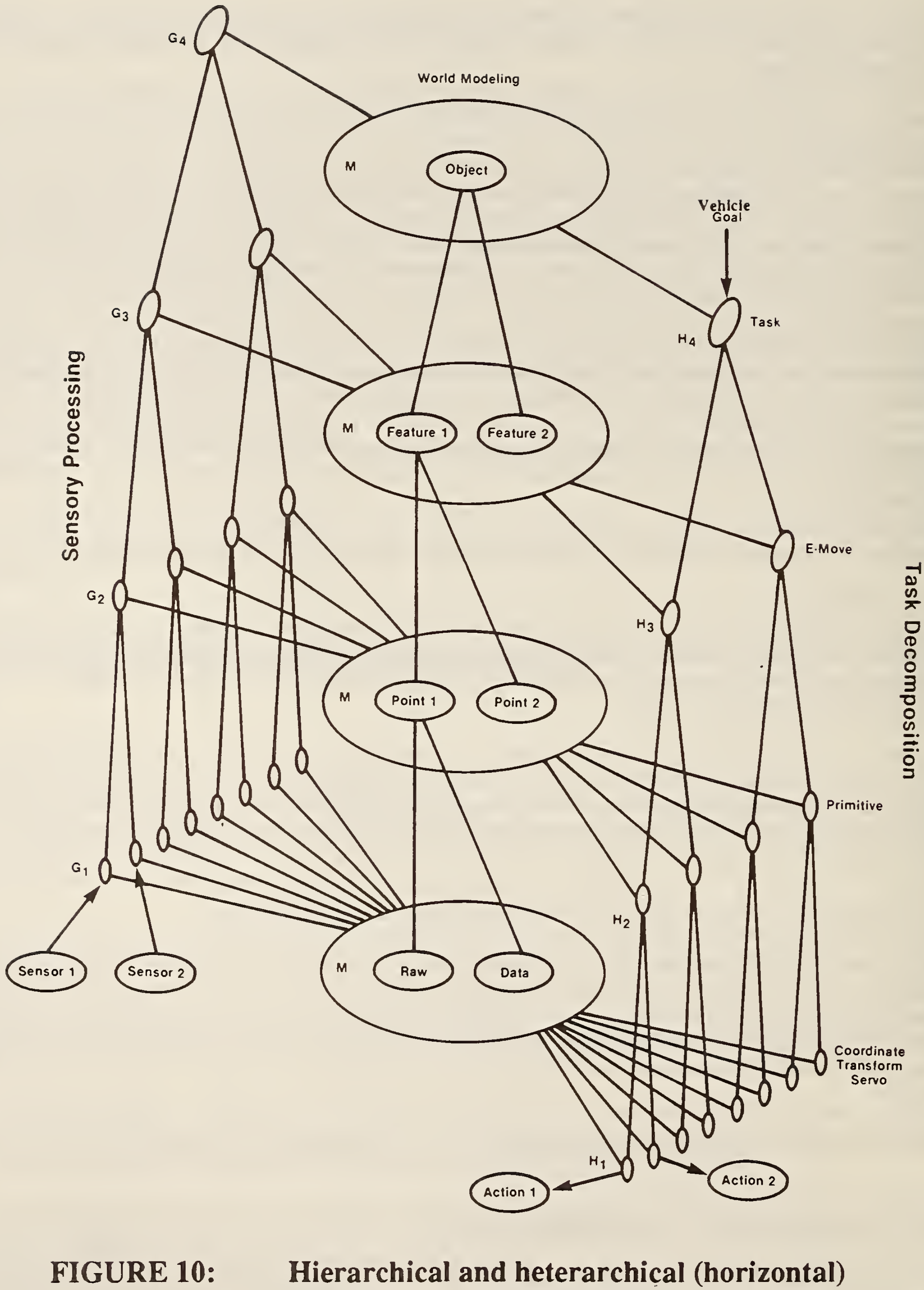


FIGURE 10: **Hierarchical and heterarchical (horizontal) organization of RCS-3 control architecture.**

Hierarchical control is an old and proven organizing concept. It has been used by military, government, and business bureaucracies for centuries. The principle is based on a partition of the problem domain, leading to an efficient division of labor, according to both spatial and temporal levels of resolution. Spatial resolution is manifested in the span of control and in the resolution of maps. Temporal resolution is manifested in terms of loop bandwidth, sampling interval, length of historical traces and future planning horizons.

Hierarchical control was applied to computer control systems for process control during the 1950s and '60s. It has been applied to computer integrated manufacturing systems during the 1970s and '80s. Real-time hierarchical control concepts are also now being implemented in advanced aircraft flight controllers and modern smart weapons systems. The concepts of hierarchical control are also now being applied in the control architectures used in a number of autonomous vehicle projects, for example, Hughes [33,34], Martin Marietta [35], Carnegie Mellon University [36], and Drexel University [37].

The hierarchical aspect of RCS-3 is most prominent in its method of decomposition of tasks, its representation of space, and its processing of sensory information. The flow of commands and status feedback is strictly hierarchical. High level commands, or goals, are decomposed both spatially and temporally through a hierarchy of control levels into strings and patterns of subcommands. Each task decomposition module represents a node in a command tree. Each command node receives input commands from one and only one supervisor, and outputs a temporal string of subcommands to one or more subordinate modules at the next level down in the tree. Outputs from the bottom level consist of drive signals to motors and actuators.

The flow of commands through the hierarchical task decomposition command tree is strictly enforced (no command subtree in RCS-3 ever reports to more than one supervisor at any instant in time). However, the RCS-3 command tree is not necessarily stationary. For example, at the Group level, the command tree may be reorganized from time to time so as to reassign vehicles to different groups for various tasks. This concept corresponds to the "virtual cell" developed by McLean [38]. When the command tree is reconfigured it is done instantaneously, and the control structure always remains a tree. No module ever has more than one superior at any one time, and all modules are always part of a command and control tree, even when one or more vehicles become separated from the others so that communication is not possible. (see Section 4.3.4) The command tree has one root node at the top, where the longest term strategy is pursued and the highest level priority is determined.

4.3.2 Heterarchical (Horizontal) Organization in RCS-3

RCS-3 is heterarchical (or horizontal) in the sense that data is shared horizonally between heterogeneous modules at the same level. At each hierarchical level, RCS-3 is horizontally partitioned into three sections: Task Decomposition, World Modeling, and Sensory Processing.

Task decomposition includes planning and task monitoring, value driven decisions, servo control, and interfaces for operator input. The task decomposition module at each level in the control hierarchy is made up of a planner manager module, plus one or more planner and executor modules. Each of these communicates voluminously with each other and with the world modeling module at the same level.

World modeling includes geometric models of objects and structures, maps of areas and volumes, lists of objects with their features and attributes, and tables of state variables which describe both the system and the environment. The world modeling module at each level consists of a set of processes that maintain maps, update lists of objects and their attributes, keep state variables current, generate predictions and compute evaluation functions based on hypothesized or planned actions. Each world modeling module is constantly in communication with the sensory processing module at its corresponding level, predicting sensory input, and being updated by the observed state of the world. Each world modeling module also responds to "What is?" and "What if?" questions from the executors and planners in the task decomposition module at its corresponding level.

Sensory processing includes signal processing, detection of patterns, recognition of features, objects, and relationships, and correlation and differencing of observations versus expectations. Each sensory processing module is made up of a heterogeneous group of processes that compute spatial and temporal correlations, differences, convolutions, and integrations; comparing predictions generated by the modeling module at the same level with observations detected by lower level sensory processing modules. The sensory processing module is programmed to filter, detect, recognize, measure, and otherwise extract from the sensory data stream the information necessary to keep the world model at its level updated.

Thus, although the RCS-3 system architecture incorporates a command and control hierarchy in the form of a formal logical tree, the horizontal communication not only exists, it predominates, both in terms of volume and bandwidth. There exists a voluminous horizontal flow of non-command information between H, M, and G modules at the same level. This information is about both the state of the task and of the world. The flow of information between sensory processing, world modeling, and task decomposition modules at each level completely dwarfs the amount flowing vertically in the command hierarchy.

The RCS-3 design is thus an attempt to take advantage of the benefits of both hierarchies and heterarchies, and to minimize the limitations of each.

It should be noted that the requirement for horizontal flow of data is mostly confined within the same subtree. The requirement for communications becomes less voluminous and critical between entities in separate subtrees. For example, horizontal communication may be important for coordinating actions between the pilot and sonar subsystems in the same vehicle. Yet the need for communication between the pilot in one vehicle and the sonar subsystem in another vehicle is limited, if not non-existent. In general, the volume and bandwidth of communication between entities at the same hierarchical level diminishes rapidly as the distance between the respective subtrees increases in the command tree.

4.3.3 Global Representation of Data

It should also be noted that RCS-3 does not restrict flow of information to only hierarchical or horizontal pathways. All input and output variables to all of the modules at all levels are globally defined, and exist in a global memory. There is no restriction prohibiting any module at any level from making a query of, or obtaining information from, any other module at any level.

RCS-3 is designed for a real-time operating system with a communications process which allows shared access to information in global memory. This communication process is transparent to the computing modules. This makes the global memory appear to the various computing modules as if it were a single common memory.

This global memory is, however, not in a single common memory. The physical architecture of global memory is distributed over a number of single board computers and memory cards. For some applications, portions may even exist as virtual memory on disks that may also be distributed in various locations. Thus, world state variables may, in practice, be distributed over a number of physically distinct memories and mass storage devices in widely separated locations. Various parts of global memory may sometimes not even be in direct communication with each other. They may, for example, be in separate vehicles.

4.3.4 Hierarchical Control of Multiple Vehicles

Coordinating behavior between intelligent vehicles with limited communication between them is a major unsolved research problem. One of the principal objectives of the MAUV project was to address this issue.

The control architecture shown in Figures 5 and 10 suggests that communications exist between all levels of the hierarchy at all times. In practice, this is often not possible, because vehicles, or groups, become separated from each other and from their chain of command.

In order to deal with this situation, each MAUV vehicle carries its own copy of the the entire RCS-3 control hierarchy, including its own complete copy of the world model and global memory. Every vehicle thus has the potential to be the command vehicle for the entire mission.

As long as there exists adequate communication between the vehicles, updates to the world model can be shared fully, and the world model of all the vehicles is kept identical.

So long as both the control system and world model of each vehicle is identical, the commands generated by the higher level control system in each vehicle will be identical to those generated by, and communicated from, the group leader vehicle. In this ideal case, communication of commands is unnecessary for coordinated behavior.

Of course, the world model of each vehicle is constantly being updated by sensory data. Events detected by one vehicle may be unnoticed by others. Once communications is limited, or the vehicles lose communication with each other, their world models will grow different due to different updates from different sensory inputs.

However, until the world models of the different vehicles diverge, coordinated behavior is possible without communicating commands. This means that communication silence can be maintained during the early phases of an engagement, up until the time when cooperative action must be taken on information that is not shared.

Keeping the world model in all the vehicles identical can require very large amounts of data to be communicated between the vehicles. In many cases, this will exceed the available communication bandwidth. Typically, the bandwidth required to transmit commands and status feedback is less than that needed to keep world models identical. Thus, the transmission of commands tends to be the mechanism of choice for coordinated behavior in most cases.

There are, however, situations where a small amount of information in the world model can produce a lengthy sequence of actions to occur. In these cases, communication of the critical world model information may be more efficient and reliable than transmitting a lengthy series of commands. This is particularly true if the series of commands must be communicated during the heat of battle, while the world model information can be transmitted before the action begins.

5. Tasks and Plans

In this section we will provide a formal definition of tasks and plans, and develop a mathematica notation for representing them.

Df: Task
A task is an activity which begins with a start-event and is directed toward a goal-event. This i illustrated in Figure 11.

Df: Goal
A goal is an event which successfully terminates a task.

Df: Command
A command is an instruction to perform a task.

A command may have the form:

DO <Task> AFTER <Start Event> UNTIL <Goal Event>

or

COMMAND := DO <Task>
WHEN (Start Event)
DO (Task)
UNTIL (Goal Event)
END-DO

Df: Plan
A plan is a set of activity-event pairs which is designed to accomplish a task and produce a goal event.

Each activity in the plan leading to the goal is a planned subtask, and the event terminating each of the planned subtasks is a subgoal. The final event in the plan is the goal event. This is illustrated i Figure 11.

For tasks that involve the cooperative action of several subsystems, a plan may consist of several concurrent strings of subtasks which collectively achieve the goal event, as shown in Figure 12.

In a plan involving concurrent activities, there may be mutual constraints. Various subtasks may require that the activities of the subsystems be coordinated. A start-event for a subtask activity in one subsystem may depend on the goal-event for a subtask activity in another subsystem. For example pointing a sonar beam at a target may not be possible until the pilot maneuvers the vehicle into the proper orientation.

At each level in the task decomposition hierarchy, there exists a command vocabulary. This consists o the set of tasks that can be decomposed by that level. For each task in the command vocabulary, there exists a task frame, such as shown in Figure 13, in which there are slots for tools or equipment required for the task, conditions that must be met before the task can begin, a statement of the goal o the task, and estimates of cost and risk in the performance of the task.

The Gantt Notation

A plan can be represented in a number of different notations. The series of actions and events illustrated in Figures 11 and 12 are the form of a Gantt chart. The Gantt chart notation explicitly

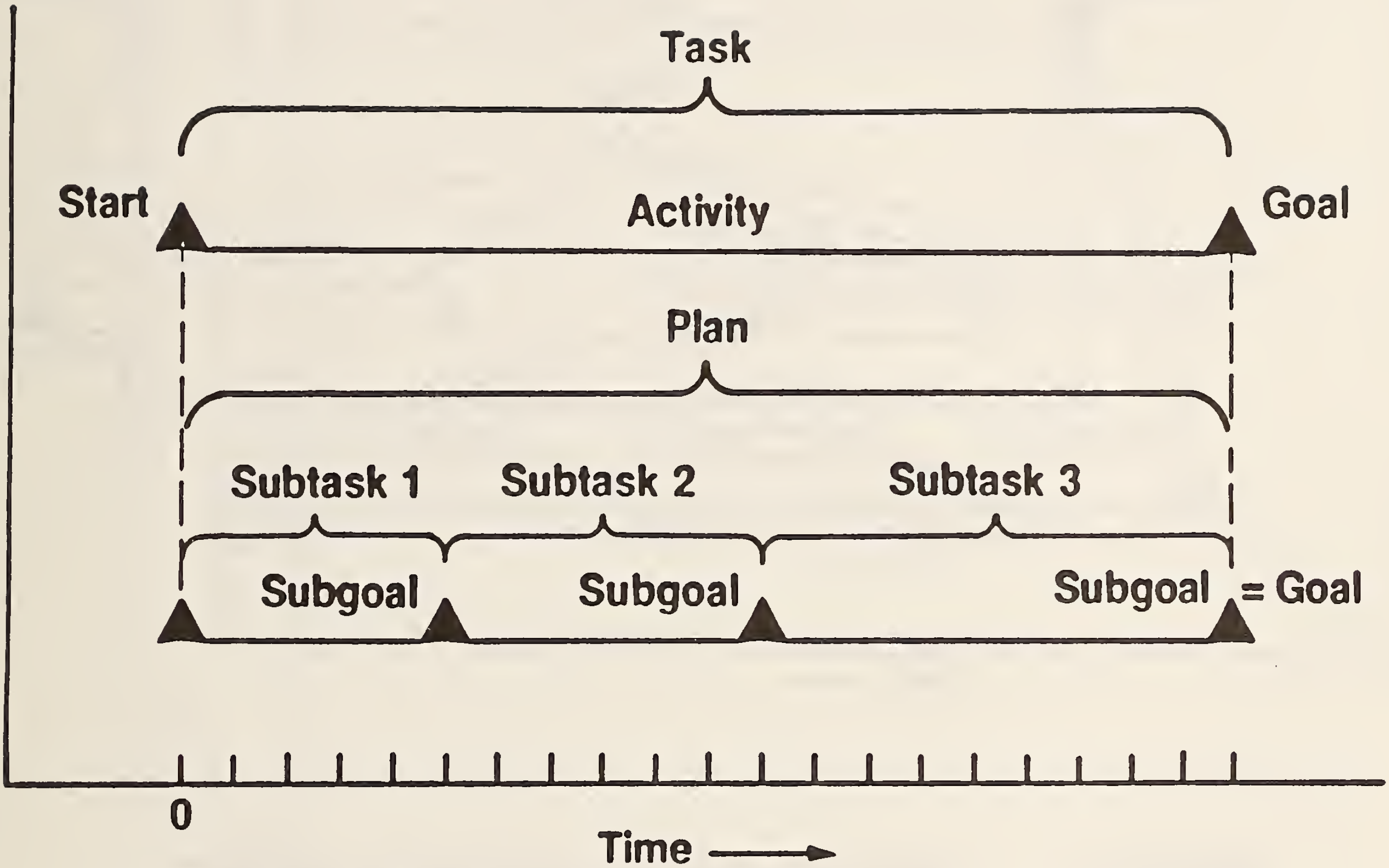


FIGURE 11: A plan is a set of activity-event pairs, or subtasks, which achieve the goal event.

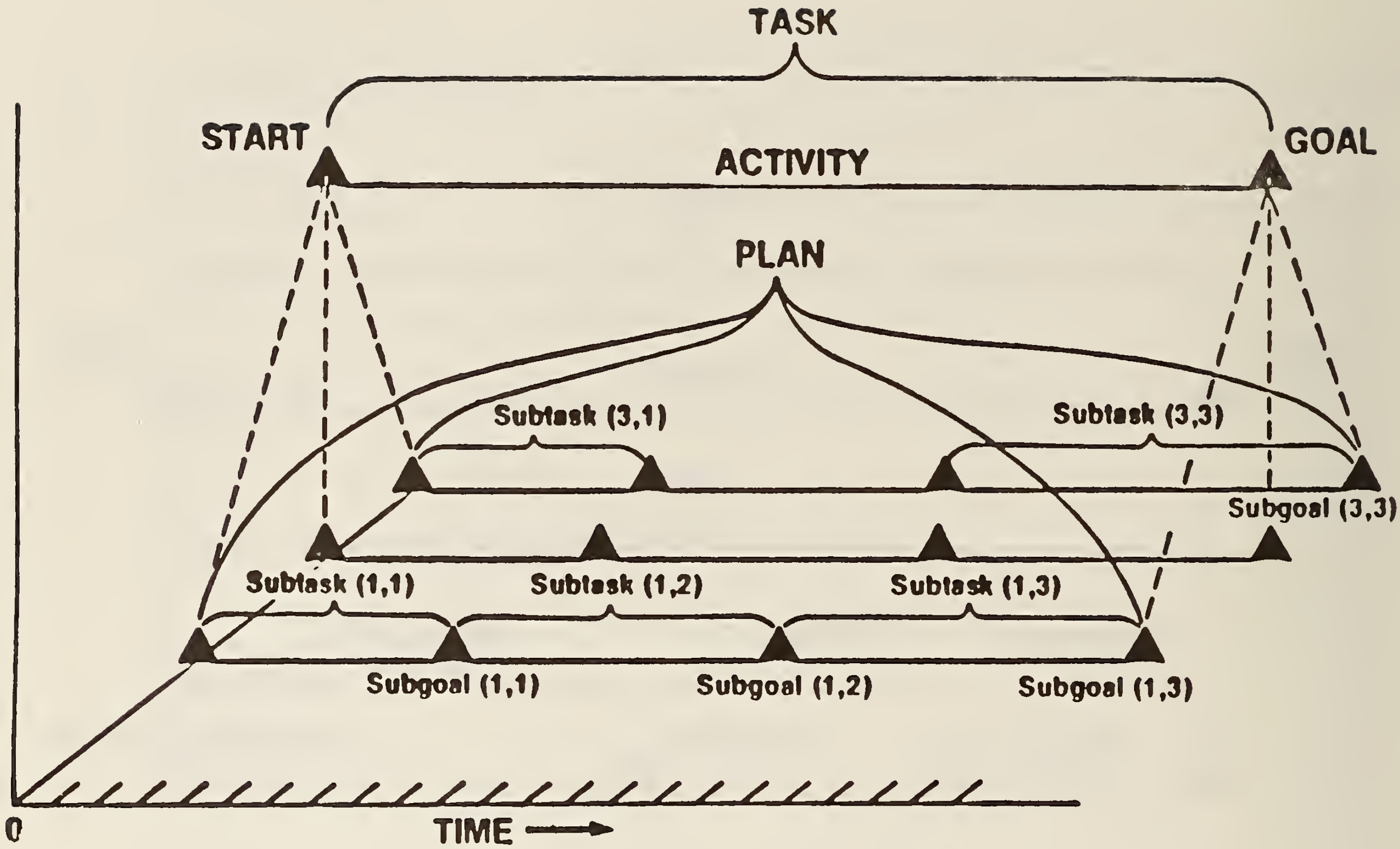


FIGURE 12: A plan may consist of several concurrent strings of subtasks which collectively achieve the goal event.

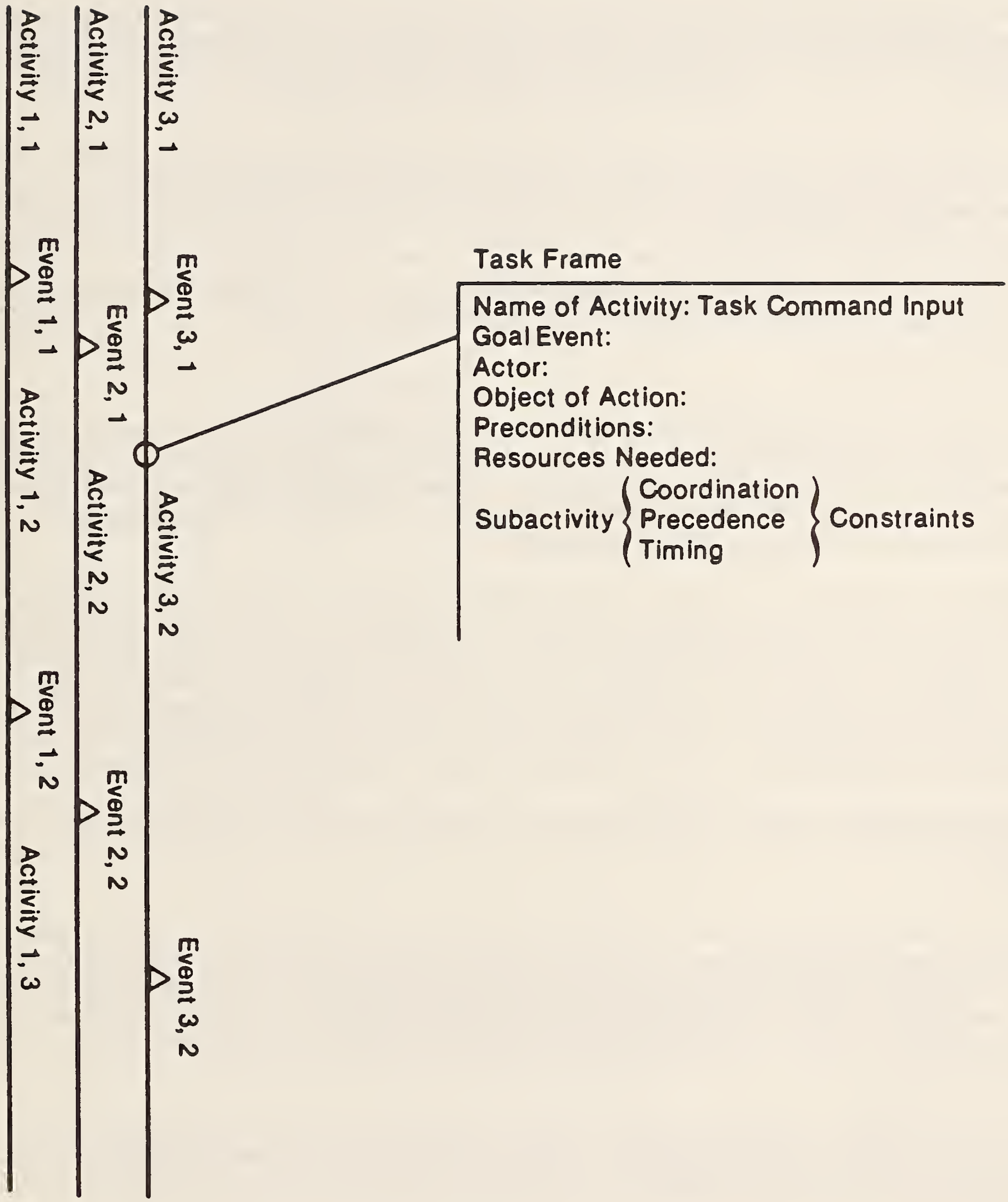


FIGURE 13. Task frame format.

represents the time axis, and can conveniently represent parallel simultaneous activities along the time axis.

However, a Gantt chart represents only one instance of a plan. If a plan is event driven, or contains conditional branches, it may have many different instances, depending on different circumstances when it is executed. The Gantt notation has no convenient way to represent a plan with conditional branching.

A Gantt chart can also be used to represent a historical trace of activities and events. For this, it is ideal, because there is only one instance of history. A historical trace can be used in two ways:
First, as a means of programming, or generating, plans. A Gantt chart of a successful sequence of subtasks can be used later as a plan.

Second, as a method for representing the processing of sensory data. The Gantt notation can be used to denote the recognition of temporal features, patterns, and events.

The State-Graph Notation

A plan can also be represented as a state-graph, as shown in Figure 14. In the state-graph notation, nodes represent actions, and edges represent events that cause one action to cease and another to begin. The state-graph notation has an advantage over the Gantt notation in that it allows steps in the plan to be event driven, and explicitly represents conditional branching. The state-graph notation is used in PERT charts, or Critical Path Method (CPM) planning charts.

A single state-graph plan may produce many different results depending on circumstances. For example, a plan containing the action node <Search Region 1> may result in finding any number of objects of interest (or possibly nothing of interest). If the plan is to do something different when different things are found, then the node in the plan graph corresponding to <Search Region 1> will have a number of edges leaving it, corresponding to the different things that might be found (including an edge for nothing being found). These different edges would then lead to different next action nodes corresponding to the different next actions that may be called for upon finding the different objects.

By defining transition edges with probabilistic conditions attached, state-graphs can be used to represent plans that involve probabilistic decision rules. This is useful in plans that implement gaming strategies.

Branching to error recovery routines at any time during a task can be handled by a slight modification to the classical state-graph formulation. A set of transition edges corresponding to error conditions can be defined as being attached to every node in the state-graph unless specifically indicated otherwise. A further modification in the traditional state-graph notation can allow counters in nodes to detect looping and generate time-out flags. This gives the plan state-graph many of the characteristics of a computer program flow-chart.

Concurrent activities of different subsystems can be represented in state-graph form by defining a separate state-graph for each subsystem. Synchronization between concurrent activities can be represented by making transition edges in one state-graph dependent on states (or transitions) in another state-graph.

Time does not appear explicitly in the state-graph notation. Time can be represented, however, by defining transition edges that depend on temporal events, such as interval time-outs or specific clock ticks. For example, time is represented in PERT charts by indicating when a node is entered and exited.

SEARCH AND ATTACK

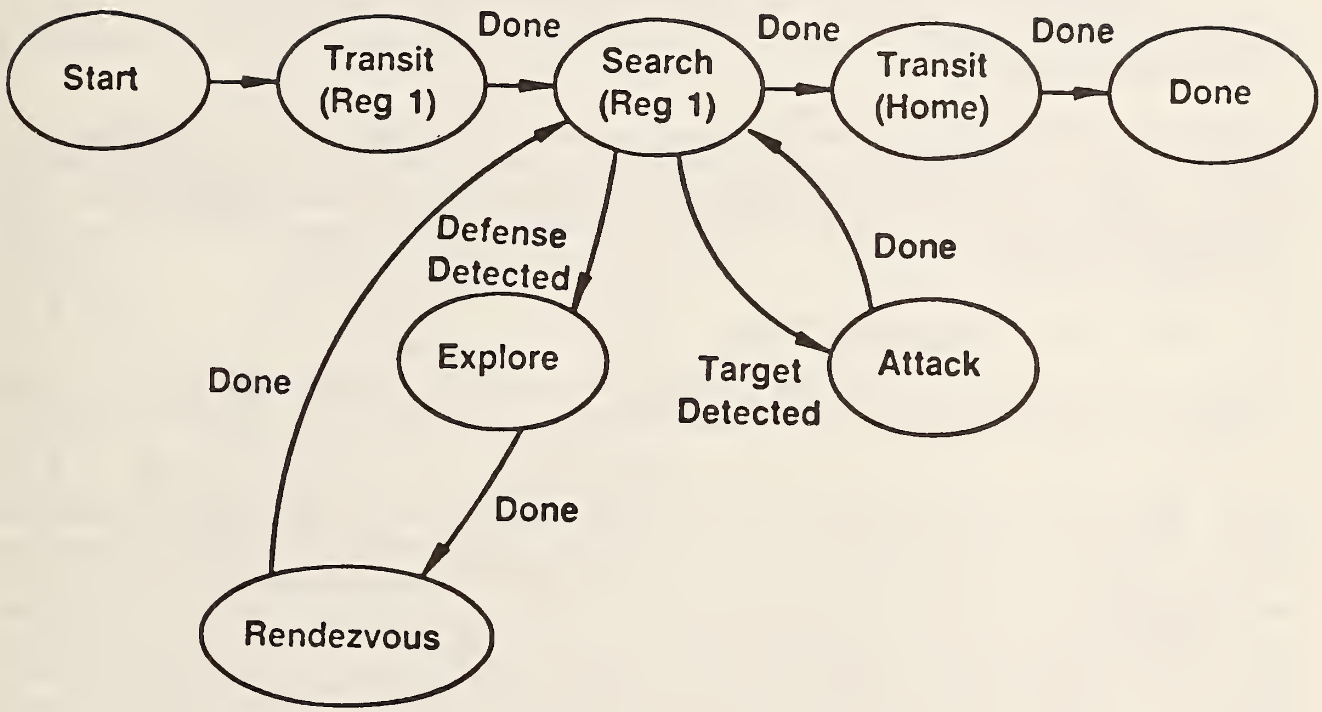


FIGURE 14. **A simple plan for a search and attack mission represented as a state graph.**

Petri nets can also be used to represent plans. Petri net plans have many of the same characteristics as state-graph plans. The principal difference between state-graph and Petri net plans is in the use of tokens, and in the correspondence between the graph and the system being modeled. In the state-graph notation, there is a separate state-graph for each subsystem, with only one token per state-graph. The position of the token is directly related to a state of the subsystem executing the plan. In contrast, a single Petri net can be used to define the activity of several subsystems. Petri net tokens also represent states, but there can be many tokens which come and go. Thus, there is no one-to-one correspondence between tokens and states of the subsystems.

The state-graph notation for a plan has been chosen for the representation of plans in RCS-3 because of the property that it can be directly translated into a state-transition table which can then be executed by a finite state automata (fsa).

At each level in the RCS-3 task decomposition hierarchy, there is a planner PL(s) which generates a plan in the form of a state-graph for each subsystem. The corresponding executor EX(s) is the fsa that executes the state transition table corresponding to that state-graph. The state-graph (or the state transition table) is thus the format of the interface between the task planners and executors. The executor fsa is defined as:

fsa = {states, transition table, inputs, outputs}.

The nodes of the plan state graph correspond to states of the fsa. Edges of the plan state graph correspond to the lines in the state-transition table of the fsa. This is illustrated in Figures 15 and 16.

Inputs consist of task commands, plan nodes corresponding to planned subtask PST(s,t), and feedback FB(s,t). Outputs are the executor outputs STX(s,t). Lines in the state-transition table also contain a pointer to the next (or same) node in the plan, a report or request other modules in the system, and possibly a pointer to a procedure to be executed when the input conditions are satisfied (see Figure 17). The procedures may be used to compute parameters (such as velocity or force) for the subtask commands. They may involve mathematical functions of time and/or state variables such as distance from target, velocity, coordinate position, etc. For example, a path trajectory procedure may compute a straight line trajectory from the current point to a goal point, or as illustrated in Figure 18, the planning procedure may compute acceleration and deceleration profiles as a function of time or position along the planned trajectory.

The state of EX(s) corresponds to the currently active node in the state graph. The output of EX(s) at time t is STX(s,t). EX(s) monitors its input PST(s,t) + FB(s,t), and discovers which line (or lines) in the fsa state-transition table match the current situation. EX(s) then executes the appropriate line in the state table; i.e. it computes the functions called, outputs the STX(s,t) subtask output commands selected, and goes to the next state node in the plan state-graph called for by that line [10, 11].

The executor fsa state-transition table has the form of a set of IF/THEN rules. Each line in the state-transition table corresponds to an IF/THEN rule for subtask selection. The state-graph form of representing plans thus can easily be translated into (or derived from) a set of expert system rules for task decomposition and subtask sequencing. The left hand side of the state-transition table corresponds to the IF premise, and the right hand side, to the THEN consequent. For example:

IF the node in the plan state-graph is PST(s,t)
and the feedback from the world model is FB(s,t)

A State-Graph Representation of Plan for Fetch (A)

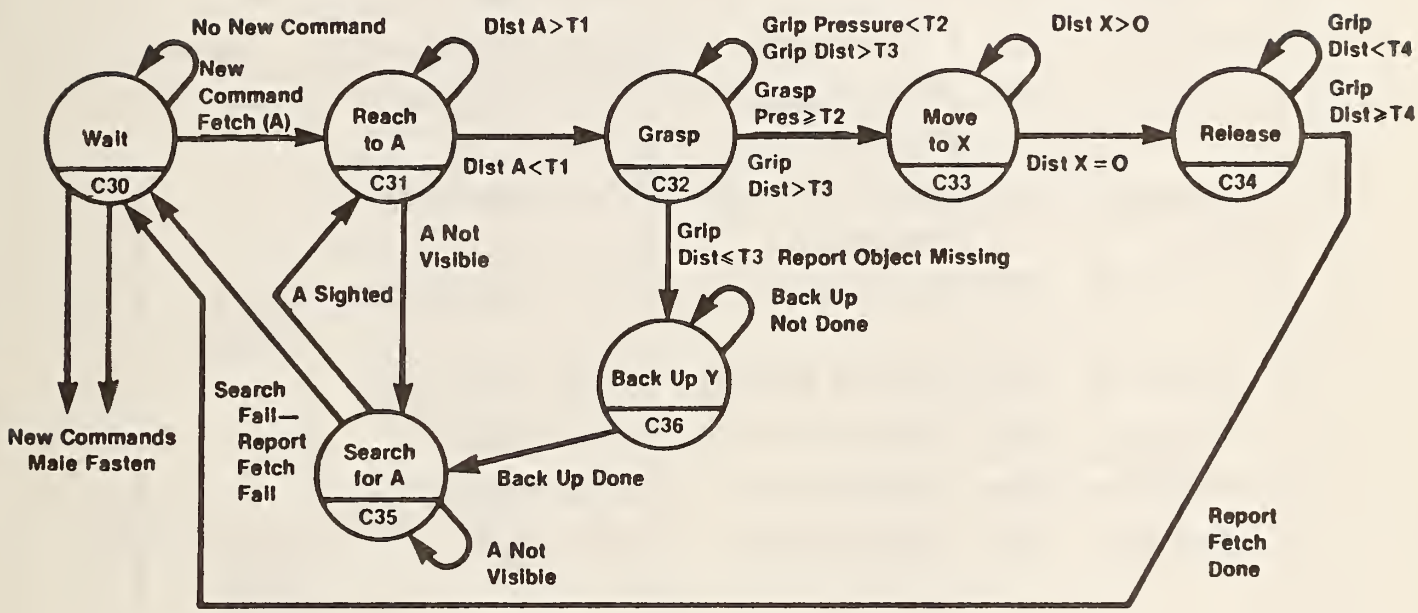


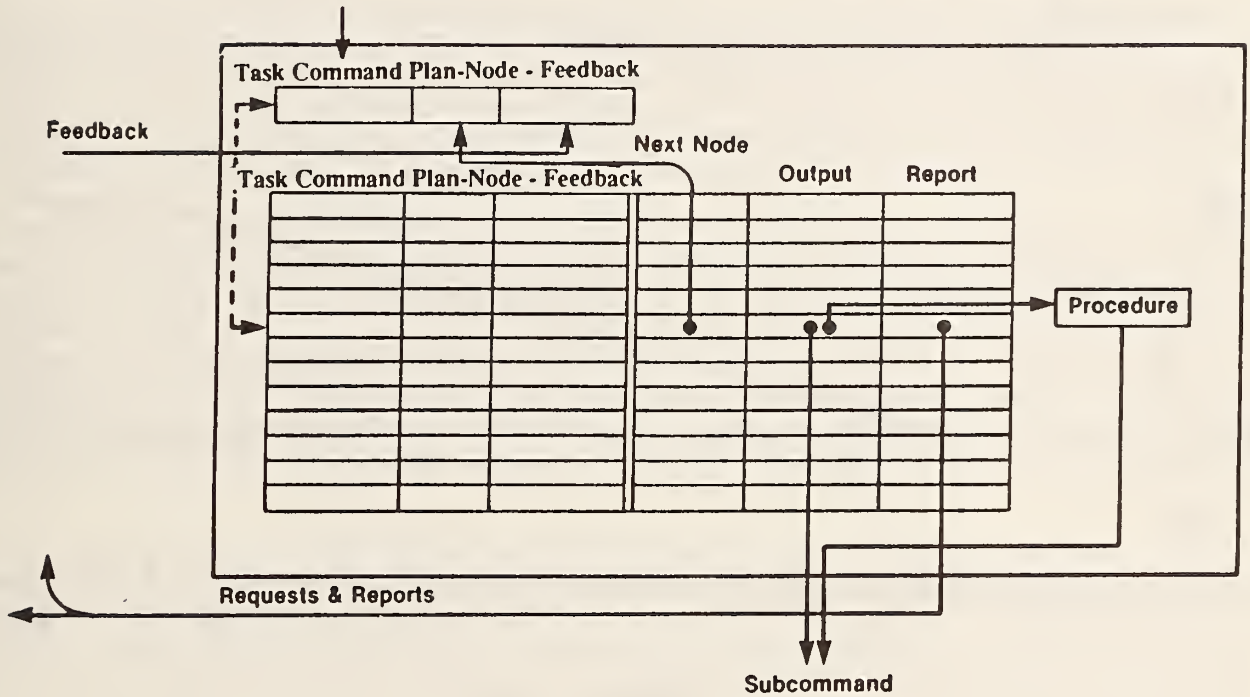
FIGURE 15. A state-graph plan for decomposition of the <Fetch (A)> command. (From reference 10)

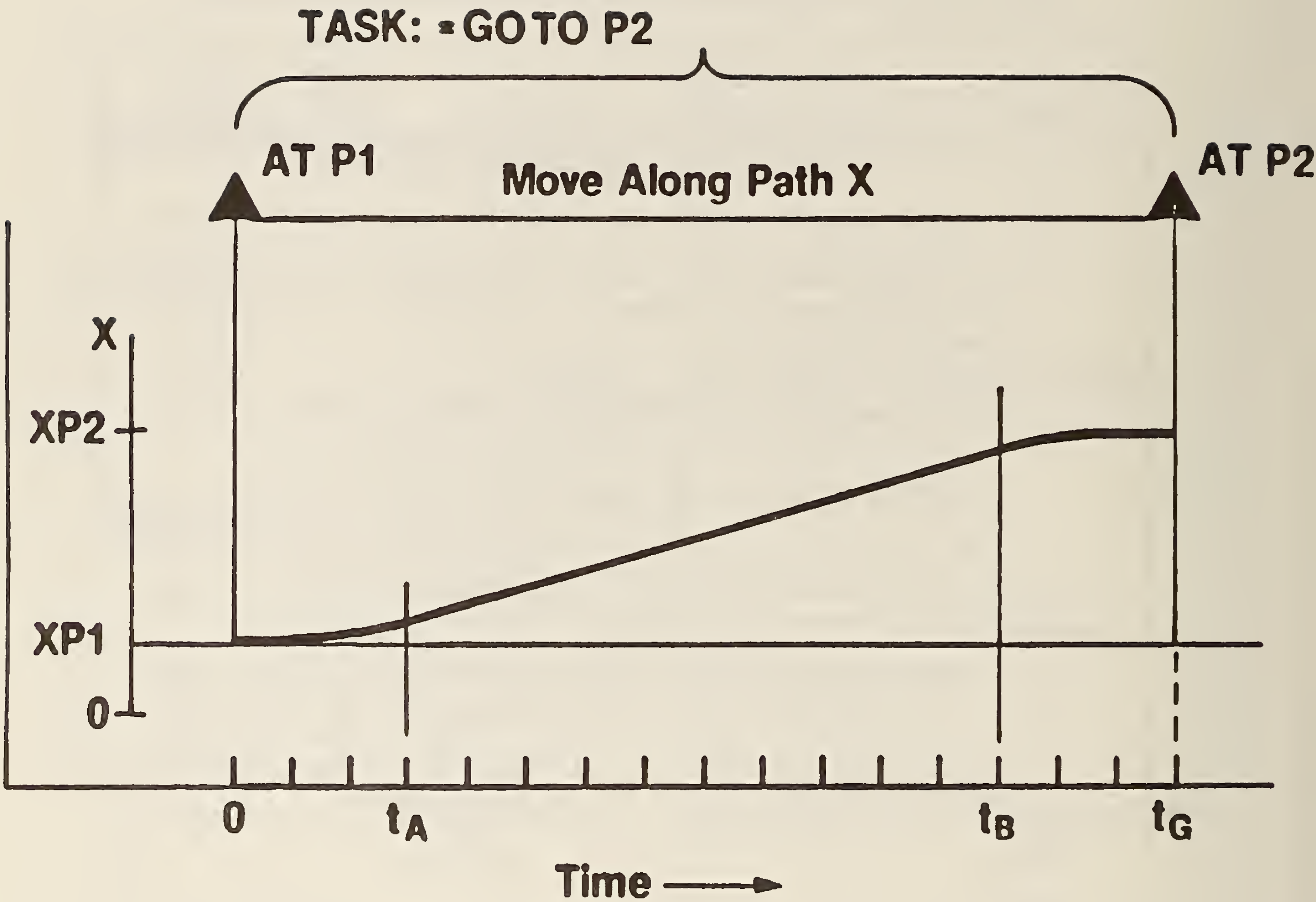
A State-Transition Table Representation of Fetch (A)

Task Command	Plan Node	Feedback	Next Node	Output	Report
—	C30	No New Command	C30	Wait	—
Fetch (A)	C30	New Command	C31	Reach to (A)	—
Fetch (A)	C31	Distance to A>T1	C31	Reach to (A)	—
Fetch (A)	C31	Distance to A⩽T1	C32	Grasp (A)	—
Fetch (A)	C31	A Not Visable	C35	Search for (A)	—
Fetch (A)	C32	Grasp Pressure<T2 Grip Dist>T3	C32	Grasp (A)	—
Fetch (A)	C32	Grasp Pressure⩾T2 Grip Dist>T3	C33	Move to (X)	—
Fetch (A)	C32	Grip Dist⩽T3	C36	Back Up (Y)	Object Missing
Fetch (A)	C33	Distance to X>O	C33	Move to (X)	—
Fetch (A)	C33	Distance to X=O	C34	Release	—
Fetch (A)	C34	Grip Dist<T4	C34	Release	—
Fetch (A)	C34	Grip Dist⩾T4	C30	Wait	Report Fetch Done
Fetch (A)	C35	A Not Visible	C35	Search for (A)	—
Fetch (A)	C35	A in Sight	C31	Reach to (A)	—
Fetch (A)	C35	Search Fail	C30	Wait	Report Fetch Fail
Fetch (A)	C36	Back Up Not Done	C36	Back Up (Y)	—
Fetch (A)	C36	Back Up Done	C35	Search for (A)	—

FIGURE 16. A state-transition table representation of the state-graph shown in Figure 14. (From reference 10)

A Computing Structure Designed to Execute State-Transition Tables





$$
X = \begin{cases} \text{for } 0 < t \leq t_A & X = XP1 + K_1 t^2 \\ \text{for } t_A < t \leq t_B & X = K_2 t + K_3 \\ \text{for } t_B < t \leq t_G & X = XP2 - K_1 (t - t_G)^2 \end{cases}
$$

FIGURE 18: An example of a path-planning procedure for moving from point p1 to p2. Only the x component of the procedure is shown.

THEN compute subtask command parameters
output subtask command STX(s,t)
report status (goal achieved?)
request feedback FB(s,t+1)
go to next (or same) node in plan state-graph

Df: Planning
Planning is the preparation of a plan.

Planning can be done off-line (long before the action begins), or in real-time (immediately before the actions begins, or as the action is proceeding). Planning may combine off-line and real-time elements. For example, off-line planning may be used to develop a library of prefabricated plans, and real-time planning can then select a particular plan, or modify a prefabricated plan in order to fit the conditions that exist at, or near, execution time. The modification of prefabricated plans can be accomplished by the procedures called by lines in the state- transition table defined by the plan graph.

Off-line planning can also be used to specify plan schemas. These are partially formed plans with prespecified constraints, such as the order that must be followed in performing certain tasks. A plan schema can be represented as a partially ordered graph, or an AND/OR graph, where nodes are actions and edges are conditions or events. Each trace through the plan schema represents the precedence constraints on a particular subtask sequence. Parallel paths formed by multiple OR edges leaving action nodes represent alternative orderings of subtask sequences. The choices among alternative traces can be determined by evaluation functions which take into account environmental conditions at, or near, execution time. Real-time planning then consists of evaluation of alternative sequences through the plan schema.

A MAUV mission will typically begin with an off-line mission plan, and prefabricated plans for all the lower levels as well. If everything goes exactly as planned, there is no need for real-time planning, or replanning. Even if there are unexpected events, the range of behavior that can be generated by a hierarchy of plan schemas, each of which contains a number of conditional branches and error recovery routines, is so large and complex that it may cover the range of situations that are likely to be encountered even in combat. If the off-line plans are sufficiently well formulated plans, with provisions for conditional branches to handle every situation that arises, and error routines to handle all emergencies, then the system will behave very intelligently and effectively without real-time planning. An efficient set of plan executors and associated parameter computation procedures is all that is needed. Only if situations arise that are not covered by existing plans, is real- time planning or replanning needed.

In general, however, it is not possible to create enough sufficiently general plan schema so that real-time planning is totally unnecessary. Military combat can be extremely unpredictable and complex. The expenditure of fuel and resources, the loss of vehicles, and fluctuations in the tide of battle may change values and priorities, and affect the choice of actions in ways that cannot be predicted before the mission begins.

The RCS-3 planners thus periodically examine the current state of the world and re-evaluate whether the current plan still gives the best mission score. If not, the current plan is replaced with the new plan giving the best mission score. A variety of real-time planning methodologies can be implemented in RCS-3. These include scripts and plan schemas; planning algorithms, which apply heuristic formulae to state variables; search methods, which hypothesize all possible actions and select the best results; and learning methods, which acquire plans from a teacher.

All these methods require the input of world model data at, or near, execution time (t=0) in order instantiate the particular plan state-graph that is to be executed in real-time. Most methods require evaluating the results of alternative plans. As shown in Figure 19, the planner may hypothesize some action or series of actions, the world model predicts the results of the action(s), and computes some evaluation function EF(s,tt) on the predicted resulting state of the world.

In the simplest case, this evaluation may be used to select between alternative AND/OR schemas, or planning algorithms, or for selecting the most effective plan state-graph for accomplishing a commanded task. In the more complex case, where an adequate plan, schema, or planning algorithm does not exist, a search method may be necessary.

The search method of planning generates a search tree, or a game tree. In the game tree, there are two types of nodes, and two types of edges. These represent the potential actions of two (or more) players in a game (or one player vs. nature).

In the case of a two player game, the first type of nodes represent states of the world prior to action by player one. The edges leaving those nodes represent alternative actions which could be taken by player one. The second type of nodes represent the state of the world after player one's action is carried out (or while it is being carried out) prior to action by player two. The second type of edges represent the set of possible actions which might be taken by player two. This is illustrated in Figure 20.

The nodes in the resulting game tree can be evaluated, or scored, based on the values and priorities assigned to objects and situations. If player two is an intelligent opponent, the probability is very high that he will always choose the move that is minimally advantageous to player one. In this case, the best planning strategy is the familiar min-max algorithm. This algorithm evaluates the state of the world for each leaf node. Then working back from each leaf: a) if a node is type one, assign to it the maximum value of all its successor nodes; b) if a node is type two, assign to it the minimum value of all its successor nodes.

The game tree then yields a plan graph by the following procedure:

Start at the root node and select the trace through the game tree which gives the maximum type two node scores and the minimum type one node scores. That trace represents the best plan. The dual of that trace is the plan state-graph, i.e.

a) For each player 1 action edge in the trace, define a plan node corresponding to the action of the edge.
b) For each type one node in the trace, define a plan edge corresponding to the condition represented by that node.

This procedure is illustrated in Figures 20 and 21.

In the case of one player against nature, the type two edge events which occur will not necessarily be the ones most disadvantageous to player one. The response of nature will be subject to some probability distribution. In this case, player one will try to maximize his score based on his best estimate of the probable future state of the environment. This he can do by multiplying the pay-off of each state of the world by the probability of that state occurring, and taking the action that leads to the highest expected score. For each probable outcome of the selected action, he plans the best next action.

In one player versus nature, the type two edges represent events of nature which might occur in response to the action of player one. Type two edges can be labeled with the probability of their occurring. In this case, the best planning strategy is to evaluate all the leaf nodes. Then working back from each leaf: a) if a node is type one, assign to it the maximum value of its successor nodes. b) if a node is type two, compute its expected value by taking the weighted sum of all its successor nodes multiplied by the probability of their occurring.

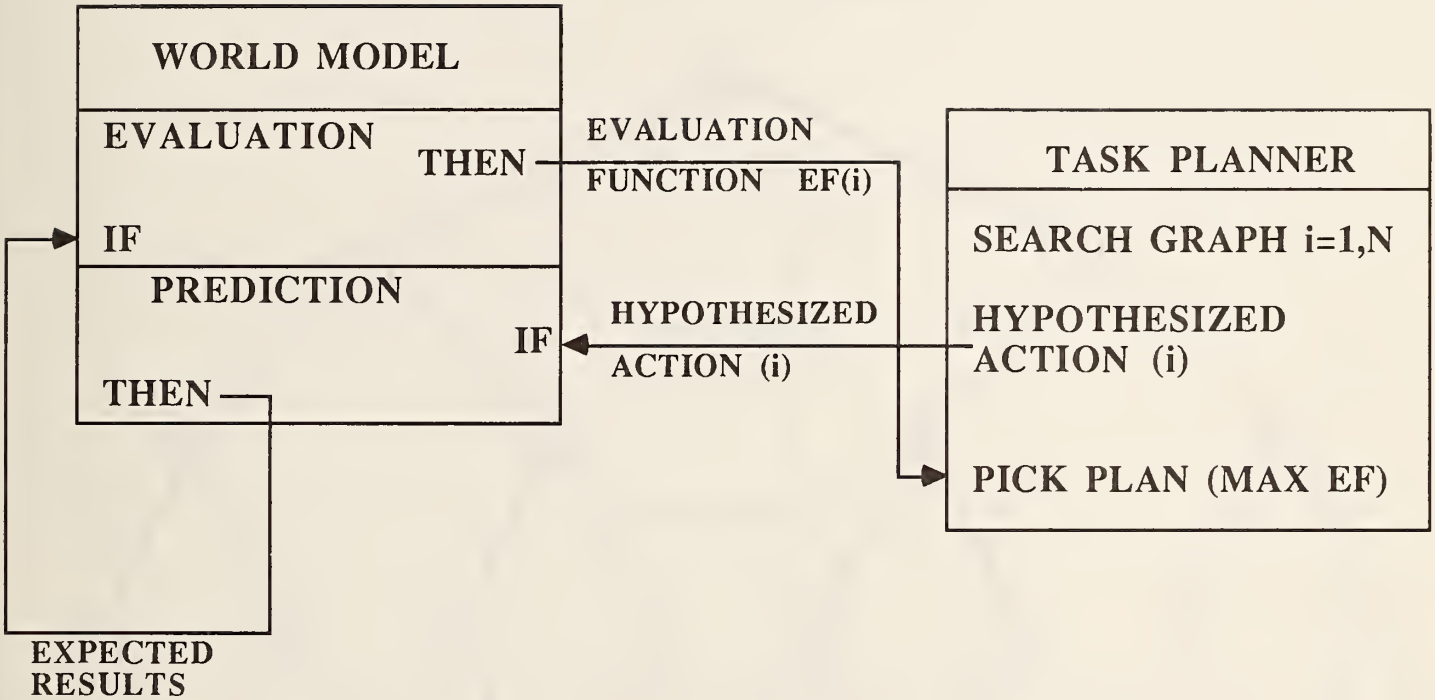


FIGURE 19. Role if world model in planning. Hypothesized actions are "What if?" questions.

GAME TREE
TWO PLAYERS

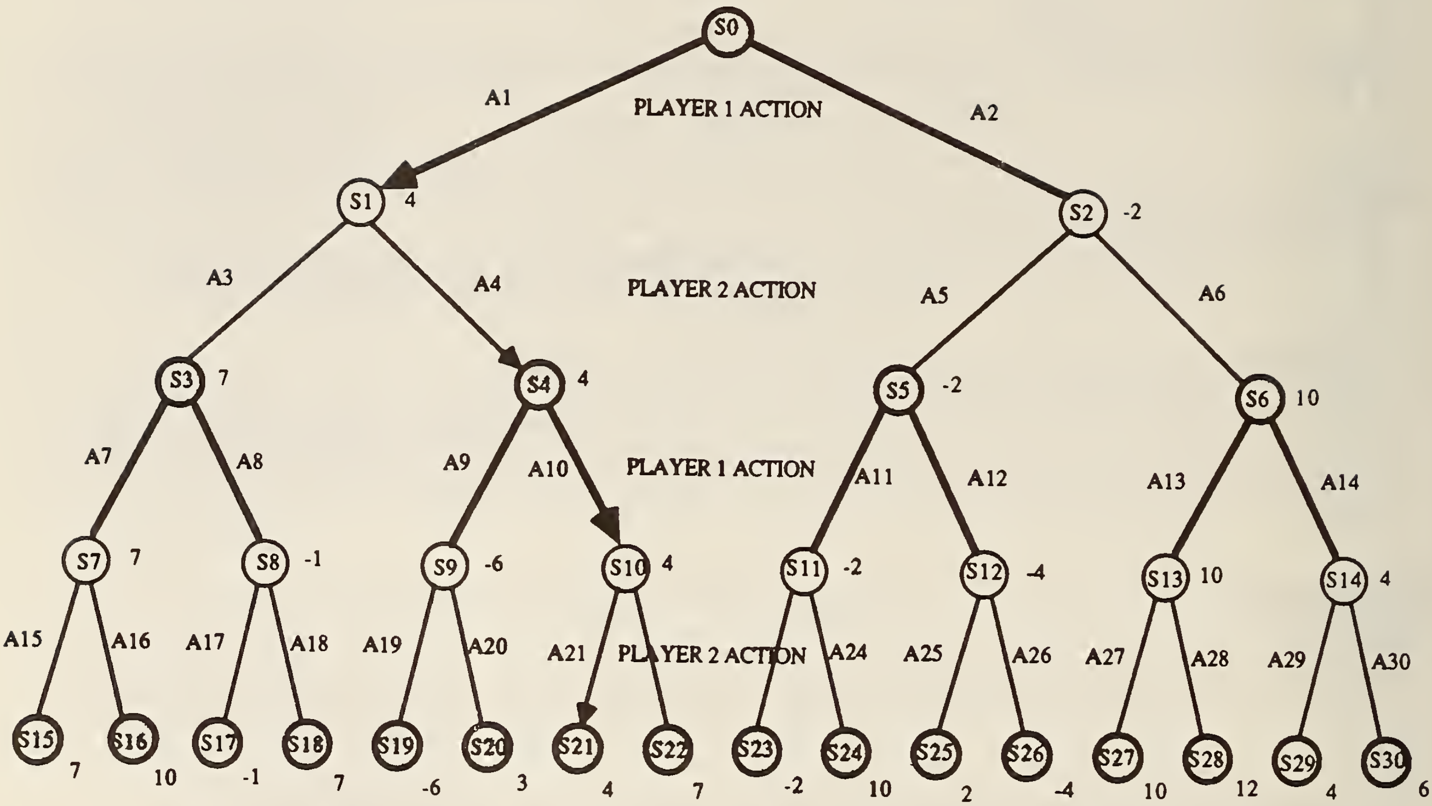


FIGURE 20. A game tree for a two player game. The nodes with heavy outlines represent states of the world prior to player one action. Values are to the right of each node.

PLAN GRAPH
TWO PLAYERS

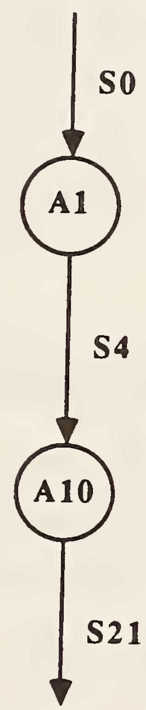


FIGURE 21. A plan graph derived from the game tree of Figure 20.

The game tree then yields a plan graph by the following procedure:
Select the traces through the game tree which give the maximum type one node scores and branch at each type two node. a) For each player 1 action edge on the trace, define a plan-graph node corresponding to that action. b) For each type one node connected to a type two node on the trace, define a plan edge corresponding to that state.

This procedure is illustrated in Figures 22 and 23.

The resulting state-graph is then the plan PST(s,tt), which can be passed to the executor EX(s) to be executed. tt is a dummy time index for steps in the plan.

Methodologies for generating plans by learning is a largely unexplored topic. However, some approaches appear promising. Perhaps the simplest is to record in Gantt chart form the actions of a human expert performing the functions of a RCS-3 planner module during the execution of a game scenario. A Gantt chart is a particular instance of a plan. It is possible to generate a Gantt chart for any particular scenario. A Gantt chart is a single trace through a plan. The Gantt chart can then be converted into a simple linear state-graph plan which can be used the next time a similar situation is encountered. Once such a linear state-graph has been generated, it can then be generalized by a human expert adding conditional branches. This can be done in a manner similar to that in which a human expert adds rules to an expert system.

Multiple scenarios will generate multiple Gantt charts, each of which is a trace through a plan schema. Methods maybe developed for building up multipath plan schemas from the systematic combination of multiple scenarios, represented by multiple Gantt charts.

Neural net mechanisms such as CMAC (Cerebellar Model Arithmetic Computer) [40, 41] also may be able to learn plans. These mechanisms not only can learn appropriate responses, but can generalize from one specific task performance to similar situations. Both learning-by-teaching, and self-learning methods are possible and appear promising.

GAME TREE
ONE PLAYER VERSUS NATURE

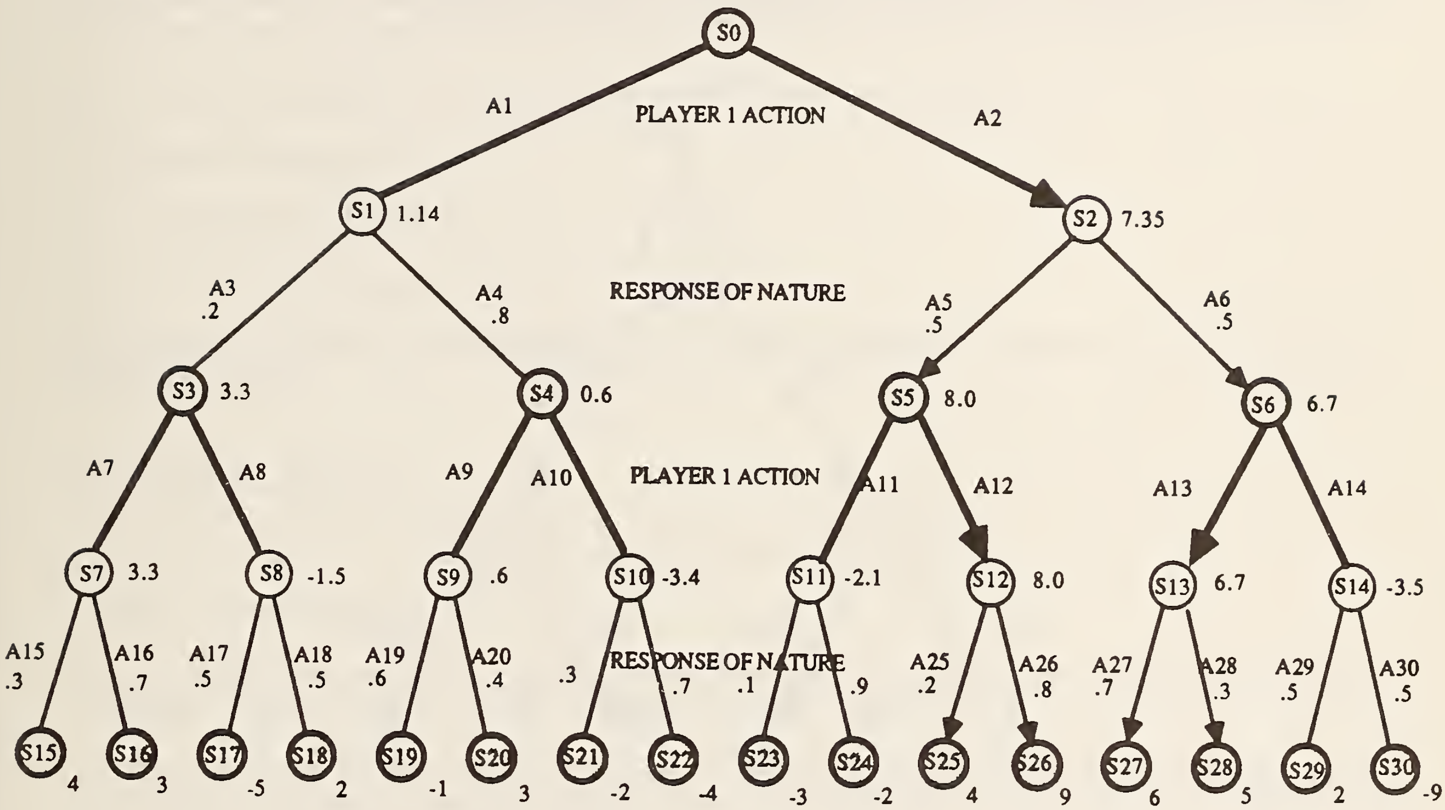


FIGURE 22. A game tree for one player versus nature. Values are to the right of each node. Probabilities of each response by nature are also indicated.

PLAN GRAPH

ONE PLAYER VERSUS NATURE

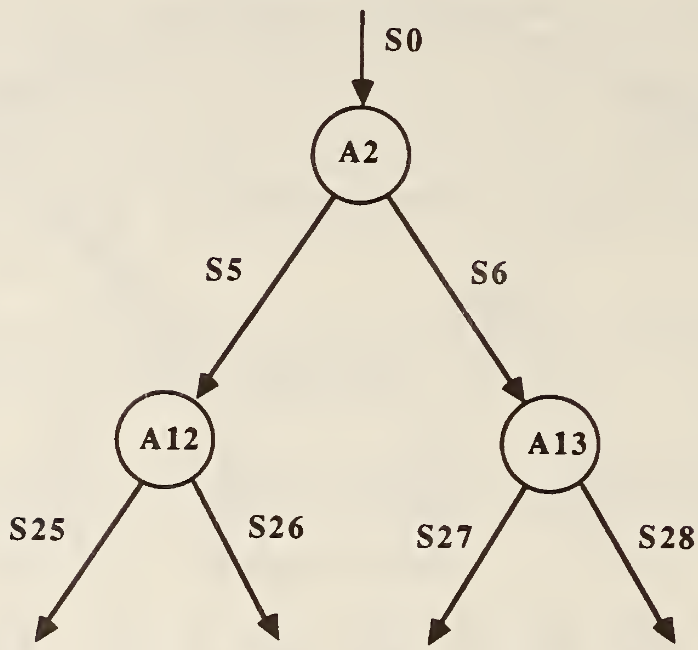


FIGURE 23. A plan graph derived from the game tree of Figure 22.

6. Task Decomposition - H modules (Plan, Execute)

The task decomposition hierarchy in Figures 5 and 9 consists of H modules which plan and execute the decomposition of high level goals into low level actions. The mission level controls several groups. The group level controls several vehicles. The vehicle level controls several vehicle subsystems. The elemental move level controls the various components of each subsystem. The primitive level controls the dynamics of each component. The servo level controls the actuators which act on the environment.

Task decomposition involves both a spatial decomposition (into concurrent actions by different subsystems), and a temporal decomposition (into sequential actions along the time line).

Each H module at each level consists of three sublevels as shown in Figure 24:

1) a planner manager PM

2) a set of planners PL(s) and

3) a set of executors EX(s).

These three sublevels decompose the input task into both spatially and temporally distinct subtasks as shown in Figure 6.

6.1 Planner Manager

As shown in Figure 25, the planner manager PM has two components:

1) A job assignment module

This module is responsible for partitioning the input task command TC into s spatially or logically distinct jobs JC(s) to be performed by s physically distinct subsystems.

At the upper levels, the job assignment modules, assign physical resources along with task elements. The output of the job assignment manager is a set of job commands JC(s), s=1, 2, ..., N where N is the number of subsystems being controlled.

2) A plan coordination module

This module is responsible for assuring that mutual constraints between subsystem plans are satisfied and that the subtasks plans for the various subsystems are coordinated where necessary.

It is the responsibility of the plan coordination module to reconcile the plans generated by each of the s planners with the plans generated by the other planners at the same level. One method is for each planner to first compute its own individual plan, and then for the planner coordinator to schedule the start and finish of the subtasks in each plan to coordinate with subtasks in other plans.

6.2 Planners

For each subsystem, there exists a planner PL(s). Each planner is responsible for decomposing the job assigned to its subsystem into a temporal sequence of planned subtasks. Each subtask has a corresponding subgoal.

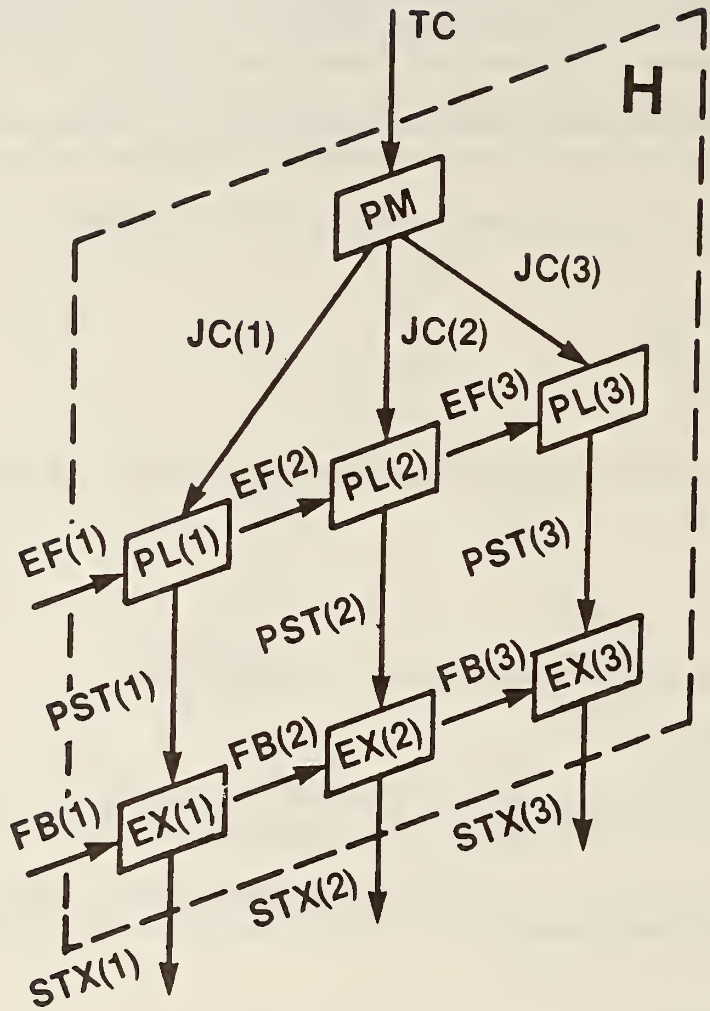


FIGURE 24. The H module at each level has three parts. A planner manager module PM, planners PL and set of executors EX.

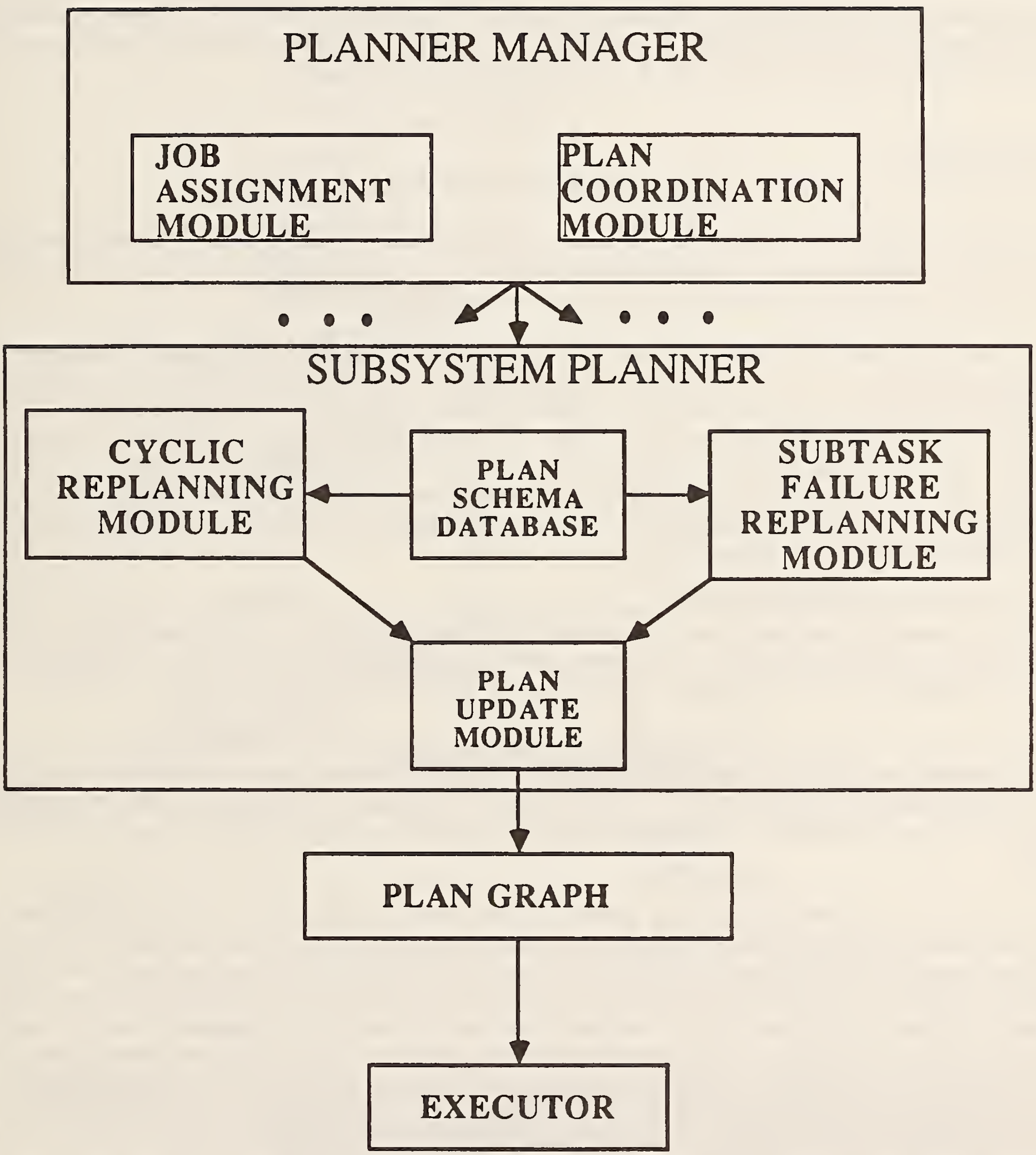


FIGURE 25. Internal structure of the planner manager and planners.

Planning typically requires evaluation of alternative hypothetical sequences of planned subtasks. Each planner PL(s) functions by hypothesizing some action or series of actions. The world model then predicts the results of the action(s) and computes the value of predicted resulting state of the world, as shown in Figure 19. This value is computed by an evaluation function which performs a priority-weighted cost-benefit analysis on the predicted results. The hypothetical sequence of actions producing the best evaluation is then selected as the plan to be executed by the executor EX(s) [42].

The representation of task planning illustrated in Figure 26 indicates that each planner generates a simple linear string of planned actions. In general, plans are more complex, with conditional branches. RCS represents plans as state-graphs which allow for conditional branches.

Df: Planning horizon
The planning horizon is the period into the future for which a plan is prepared.

Each level of the hierarchy has a planning horizon of approximately two input task time durations. This implies that the planner at each level generates a plan for the current and the next planned input task. Planning is performed top-down, and there always exists a hierarchy of plans.

Figure 27 shows a timing diagram for the RCS-3 task decomposition and sensory processing system as was tp be implemented for the the MAUV control system. The highest level input command is to accomplish the mission. The mission plan covers the entire backlog of work to be done, and the planning horizon of the mission level is the end of the mission. At each lower level, plans are formulated (or selected) in real-time to accomplish the current and next task in the plan of the level immediately above. Each task in the higher level plan is decomposed into a lower level plan of at least two, and typically less than ten, subtasks. The planning horizon thus shrinks exponentially at each successively lower level of the hierarchy.

Similarly, the rate of subtask completion, and hence the rate of subgoal events, increases at the lower levels of the hierarchy, and decreases at upper levels of the hierarchy. If the planners at each level generate plans containing an average of five steps, the average period between changes in output at each level will increase by a factor of about five at each higher level in the control hierarchy.

Replanning is done either at cyclic intervals, or whenever emergency conditions arise. The cyclic replanning interval is about an order of magnitude less than the planning horizon (or about equal to the expected output subtask time duration). Thus the real-time planner must work an order of magnitude faster than real time. Emergency replanning begins immediately upon the detection of an emergency condition.

Figure 28 shows three levels of planning activity. The activity represented by the Gantt chart at the highest level is input to the top level H module as a task command. This task is decomposed by the job assignment manager and three planners of the top H module into three simultaneous plans consisting of four activity-event pairs each. The first executor of the top level H module outputs the current subtask command in its plan to a second level H module. This second level task command is decomposed by the job assignment manager and three planners in the second level H module into three plans, again consisting of four subtasks each. The first of the second level executors outputs the current activity in its plan to a third level H module, which further decomposes it into three plans of four subtasks. At each level the final subgoal events in the plans correspond to the goal of the input task. At each successively lower level, the planning horizon becomes shorter, and the subtasks become more detailed and fine structured.

The timing diagram in Figure 27 illustrates the duality between the task decomposition and the sensory processing hierarchies. A sensory event at one hierarchical level can be defined as a sequence of events

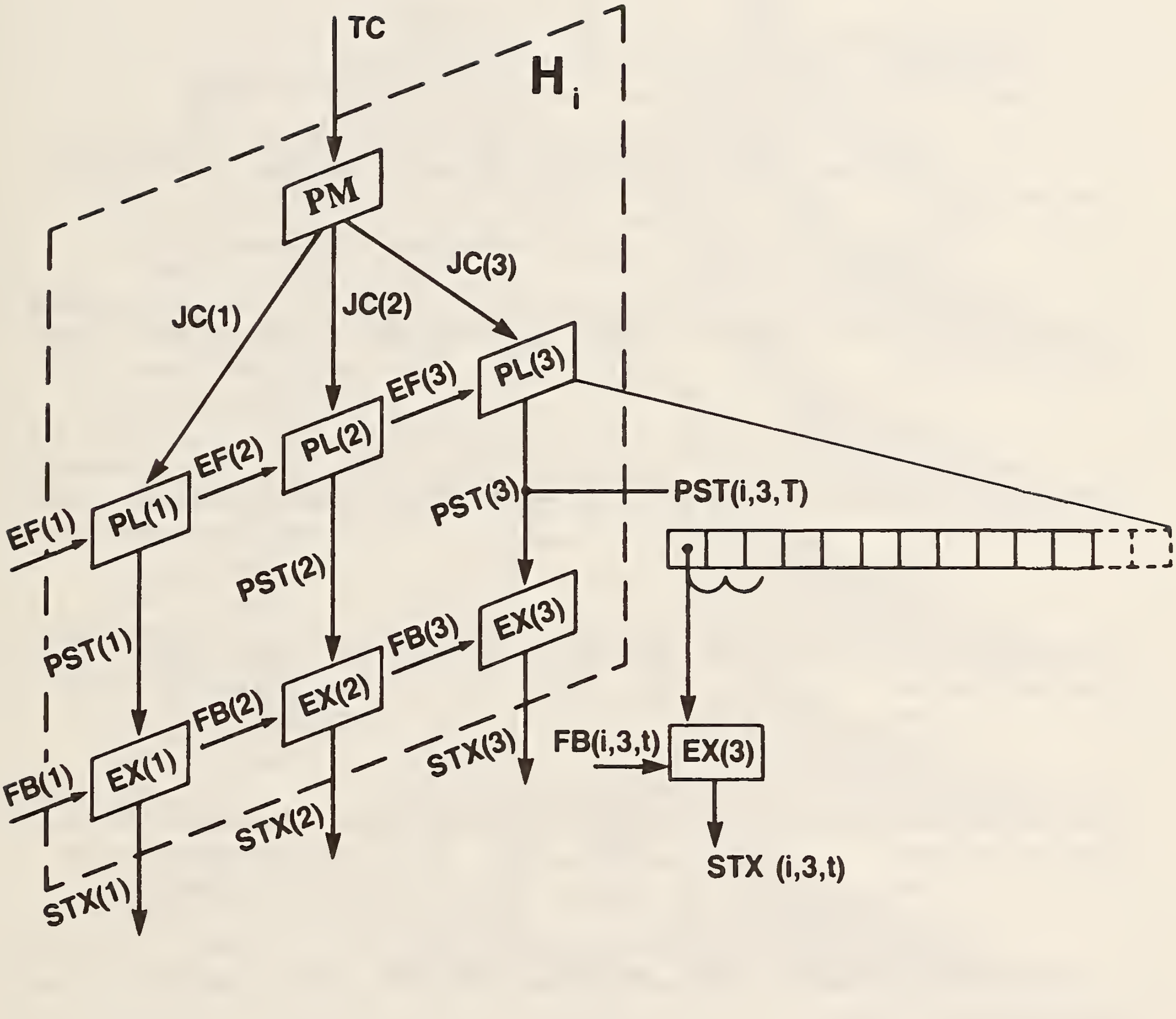


FIGURE 26. At each level i, each planner PL(j) produces a string of planned subtasks PST(i,j,t). At time t the executor EX(j) reads the planned task PST(i,j,t). The feedback FB(i,j,t) and computes an output STX(i,j,t).

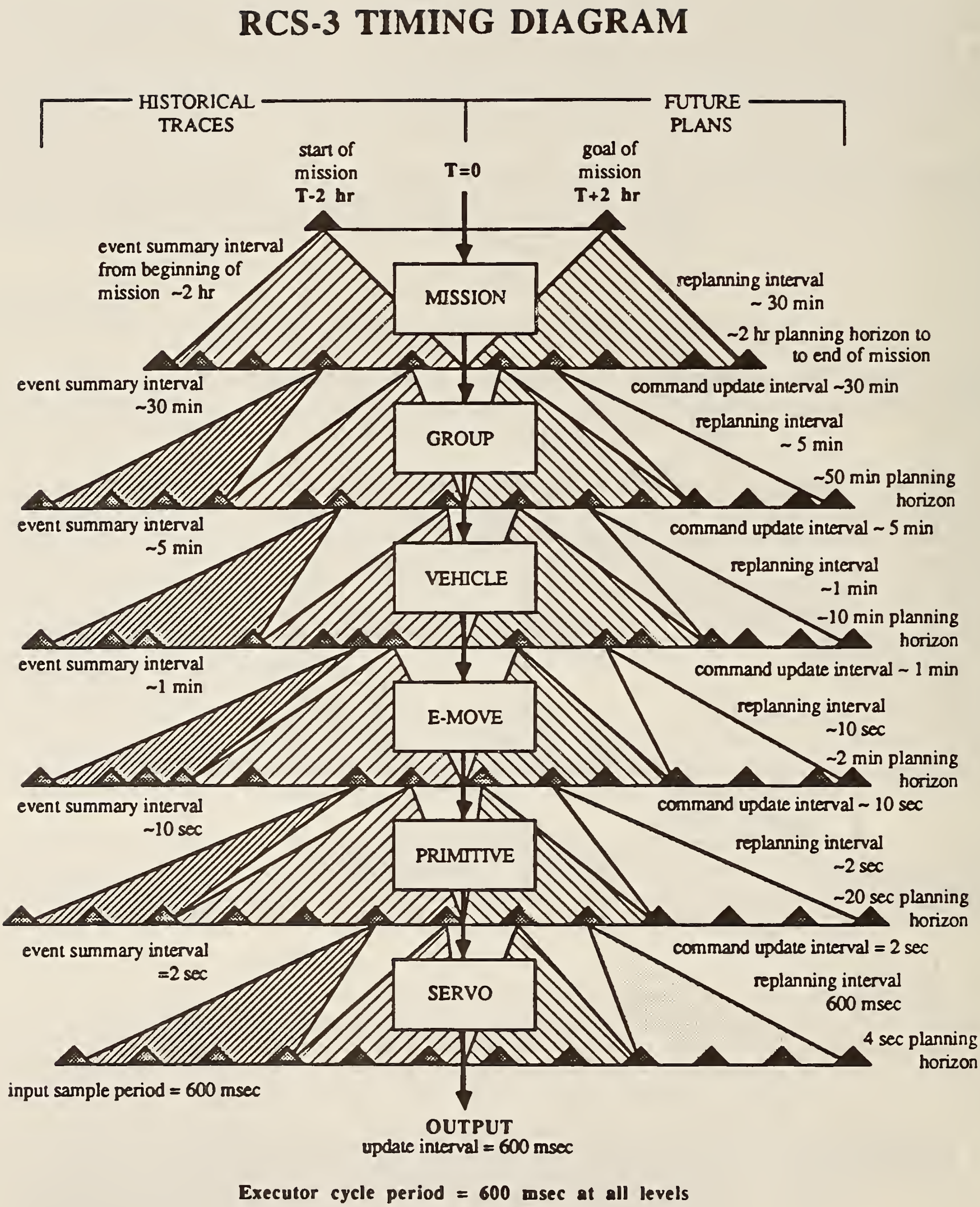


FIGURE 27. A timing diagram for the MAUV version of RCS-3 illustrating the planning and sensory processing time scales at each level.

Hierarchical Planning

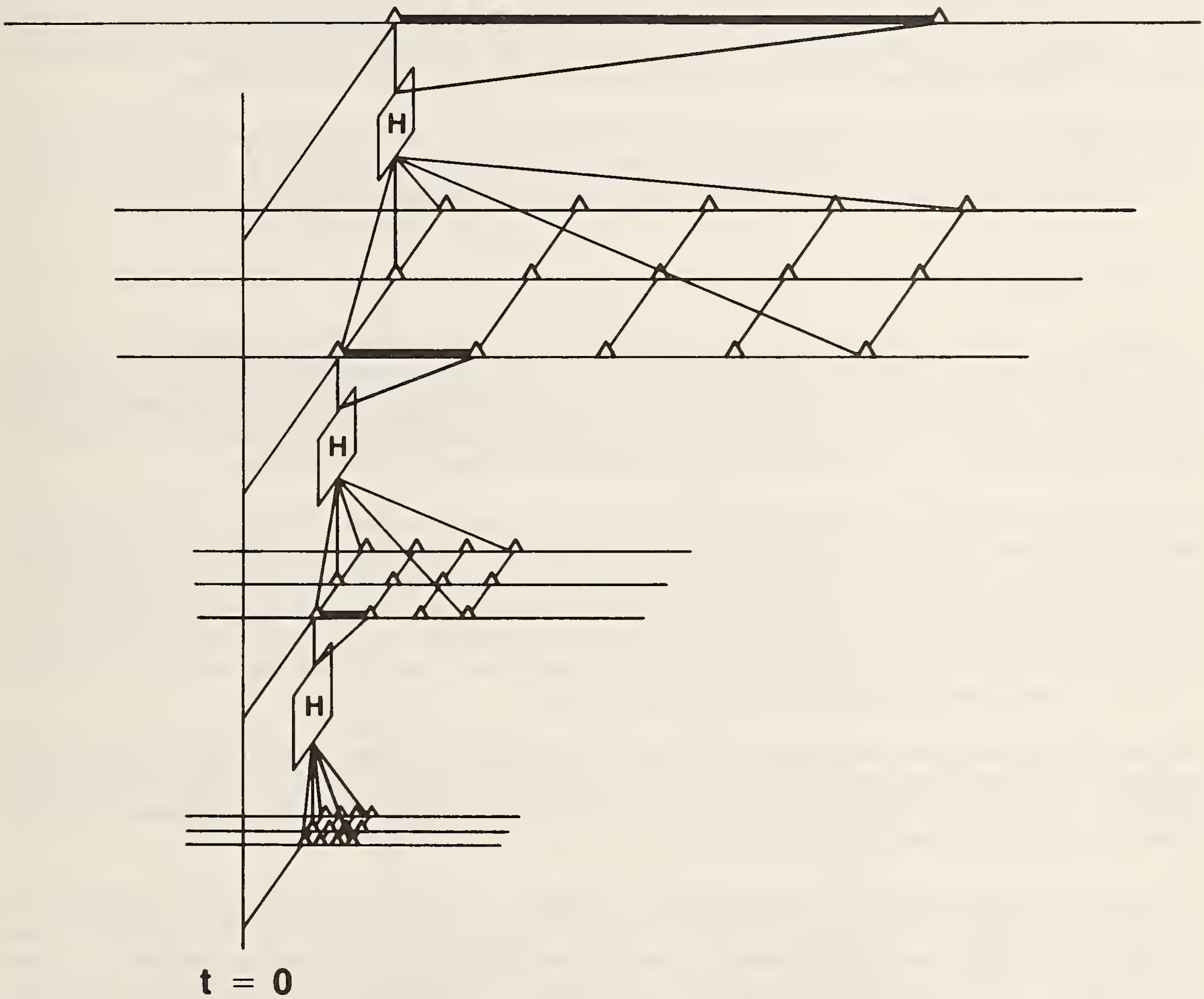


FIGURE 28. Three levels of planning activity in RCS-3.

at the next lower level. At each level in the hierarchy, the sensory processing modules look back into the past about as far as the planner modules look forward into the future. At each level, future plans have about the same detail as historical traces.

The goal events which terminate each subtask in the plan, when achieved at time t=0, become the observed events that make up the historical trace. To the extent that a historical trace is but a time shifted duplicate of a former future plan, the plan was followed and every task was accomplished as planned. To the extent that a historical trace deviates from the plan, there were surprises.

This suggests a measure of performance for robot planners. A metric which quantifies the extent to which the historical trace deviates from the plan could be integrated over the period of a task. The inverse of the resulting value would provide a figure of merit for a robot planner.

6.3 Executors

For each planner PL(s), there is an executor EX(s) which is responsible for successfully executing the plan prepared by its respective planner. When each subtask in the current plan is successfully completed, the executor steps to the next planned subtask. When all the subtasks in the current plan are successfully executed, (i.e. when all the subgoals in the plan are successfully achieved), then the goal of the plan is achieved. The executor then steps to the first subtask in the next plan.

The executor modules operate on short, regular intervals, or execution cycles. A flow chart of the executor is shown in Figure 29. The length of the execution cycle is set by a system state clock. The period of the state clock is defined by the rate at which sensory input data is sampled. In the MAUV control system, the executor state clock at all levels increments every 600 milliseconds. Other implementations of RCS-3, may use other time increments. For example, the NASREM [6] implementation for the space station telerobot manipulator uses a one millisecond executor cycle at the servo level, and submultiples of this rate at higher levels.

The executor at each level has the task of reacting to feedback in one state clock period. If the feedback indicates the failure of a planned subtask, the executor branches immediately to a preplanned emergency subtask. The planner simultaneously selects or generates an error recovery sequence which it substitutes for the former plan which failed.

If unexpected events cause a plan to become obsolete, and if no error recovery procedures or emergency subtask is adequate to deal with the current situation, the control system is without a plan. A condition in which one or more levels has no plan available for execution can be described as a state of "confusion". The time required to generate a new plan is an important system parameter, and what the system does while a new plan is being computed is an important issue in error recovery and restart.

Every time the state clock increments one count, the executor executes a communicate-compute-wait sequence as shown in Figure 30.

COMMUNICATE

During the communicate interval, the operating system moves data from process output buffers to process input buffers. This can be done either by actually moving data, or by changing pointers. It also updates world model global data variables.

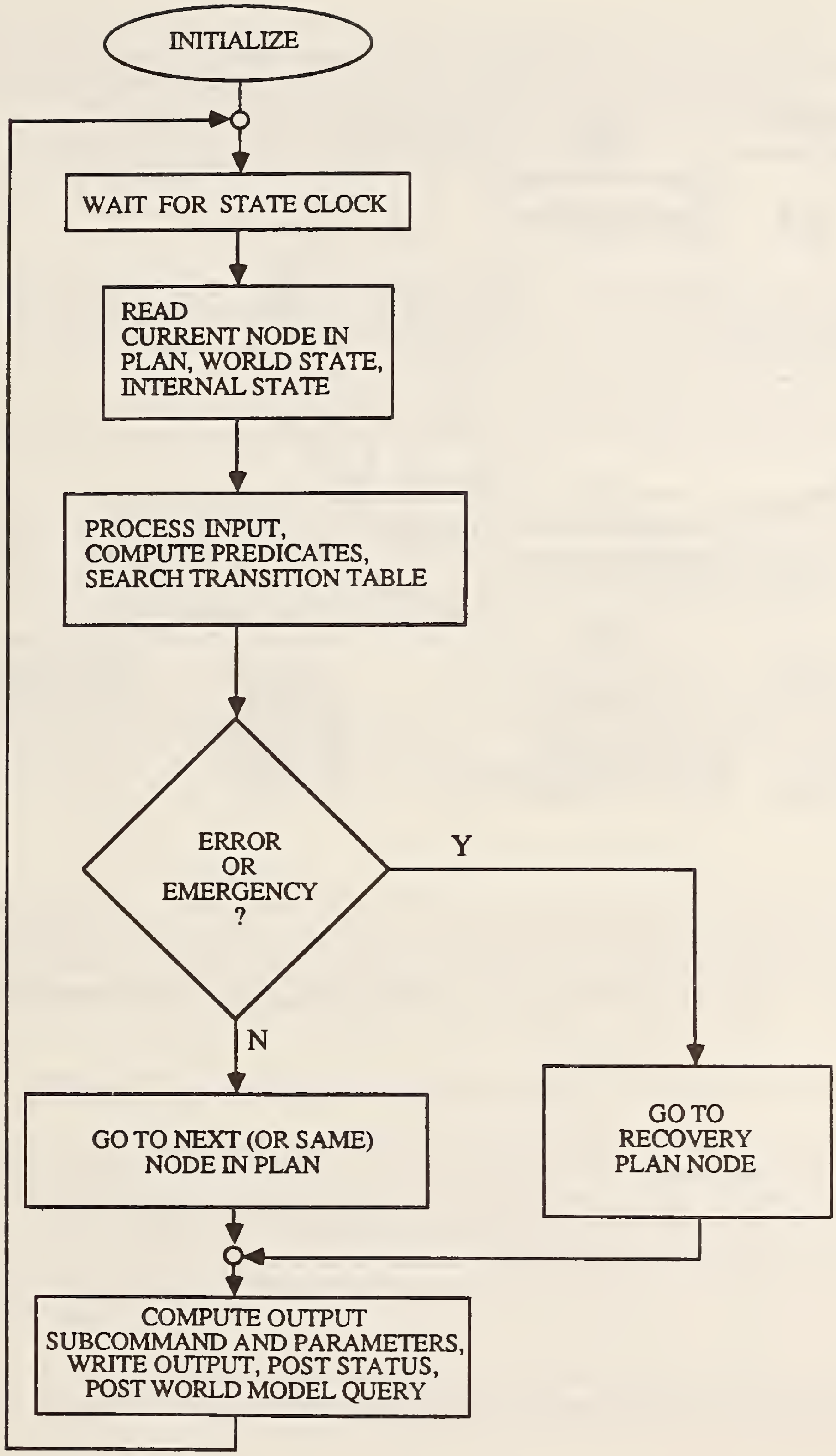


FIGURE 29. **A flow chart of the executor modules at each level.**

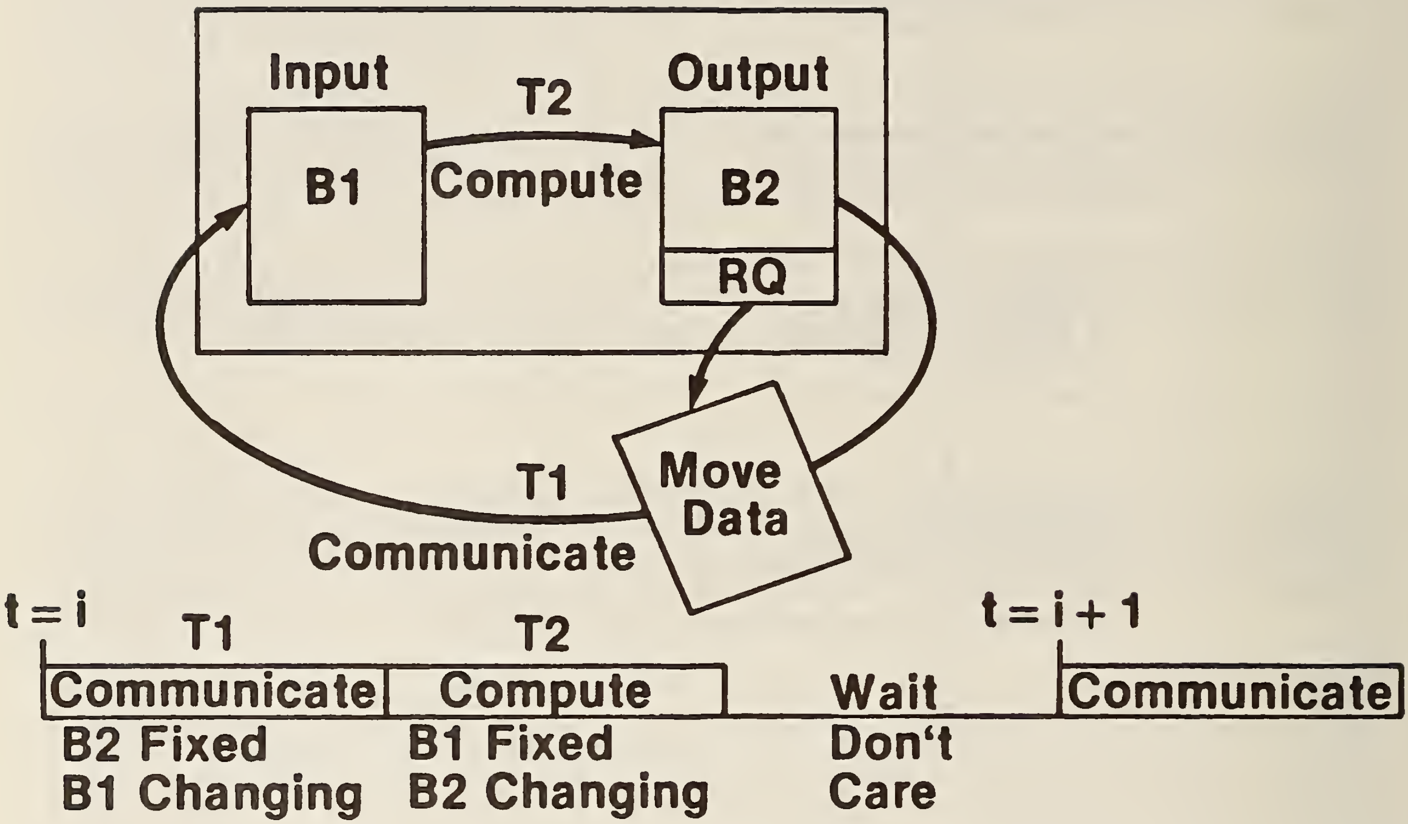


FIGURE 30. Executor timing for communicate-compute-wait cycle.

At the beginning of the communicate interval each executor reads:

a) the current subtask in the plan generated by its respective planner
b) feedback from the world model reporting current state of the world
c) status from the planners/executors at the next lower level
d) status from the other executors at the same level

COMPUTE

During the compute interval, the various executors access input buffers and global variables in the world model. Each executor (possibly in parallel) performs a number of calculations. These include processing the input, if necessary, to put it into the proper form for computing predicates.

Each executor then searches its list of predicates and computes whether any conditions are satisfied that would cause it to step to another state in the plan graph. If not, the executor stays in the current state.

Each executor then computes an output subcommand to the next lower level in the control hierarchy. This output may be simply a symbolic subcommand stored in the current node of the plan graph, or it may contain numerical parameters that depend on command and feedback variables. The output parameters may, for example, be computed by an algorithm which compares the planned subtask goal with the state of the world reported by feedback, and generates an output designed to null the difference between the current state and the goal state. In this case the executor acts as a servo, closing a control loop at its particular level of the hierarchy.

Finally during the compute interval, each executor writes an output subcommand into its output command buffer, posts a request for input from the world model in the request buffer, and puts status reports to the next higher level and to the world model in the status buffer.

WAIT

During the wait interval, the executors wait for the next increment of the state clock. Any process which finishes before the end of the compute interval, waits for the next communicate interval for new input data. Any process not finishing before the end of the compute interval continues processing until finished, and then waits for the next communicate interval for its results to be transmitted and new data acquired.

In the MAUV version of RCS-3, this communicate-compute-wait cycle repeats every 600 milliseconds.

Feedback from the world model keeps the executors informed as to events in the world. Status reports inform the executors of the state of the rest of the control system. Status reports from the next lower level provide a handshaking acknowledgment of receipt of the subtask command and an echo of the unique identification number of the command currently being executed in the next lower level. This enables each EX(s) process to know that its subtask output has been received and is being executed. Error status reports are posted if there are failures in handshaking, or if time limits for subcommand execution are exceeded.

An executor may also use feedback or status reports for coordinating its output with other executors at the same level. Coordination can be based either on the detection of events in the world, or on clock timing.

The data buffers forming the input and output buffers to an H module at the i-th level are shown in Figure 31.

The executors at all levels produce an output every 600 milliseconds. Thus, a subtask at any level can

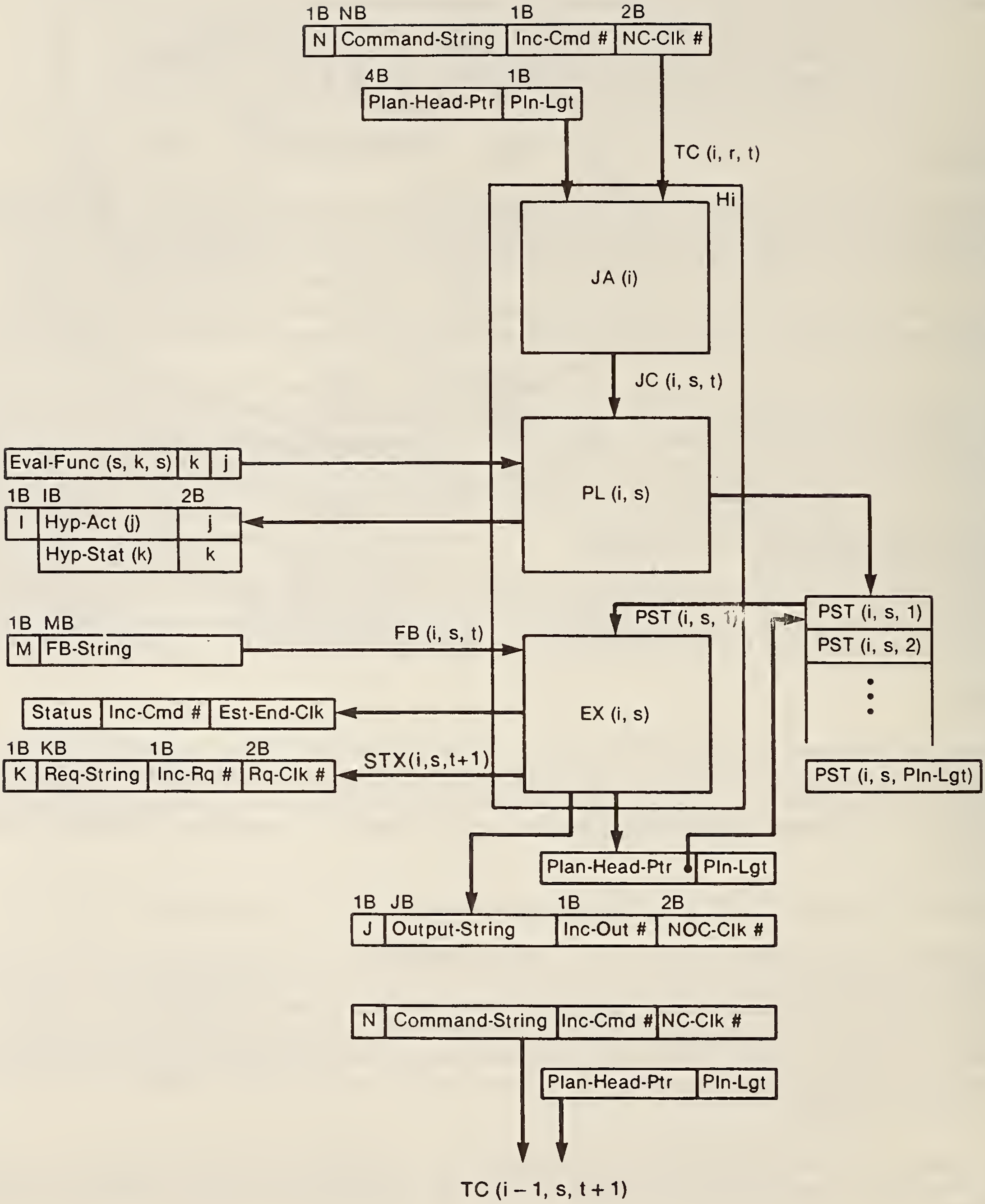


FIGURE 31. **Data buffers for input and output to the H module at the i - th level.**

be altered on any state clock cycle, and the minimum subtask period at all levels is 600 milliseconds. In other words, the finite state automata comprising the executor at each level has a state clock with 600 millisecond period, and any state in the plan state graph will be occupied by the active token for at least 600 milliseconds once it is entered.

The executor outputs typically do not change in value every 600 milliseconds, except at the servo level where 600 milliseconds is the servo sample period. The primitive level output changes every 2 seconds. At higher levels, changes in output are event driven at irregular intervals. The E-move output changes with events which occur approximately every 10 seconds. The vehicle level output changes on average every minute. The group level output changes about once every 5 minutes, and the mission level output averages about one change every 30 minutes.

In the current MAUV version of RCS-3, the primitive and servo levels reside in the University of New Hampshire controller. The E-move executor provides a framemod command to the UNH controller every 600 milliseconds. The E-move framemod output value changes approximately every 10 seconds.

A summary of the RCS-3 timing is given in the following table:

TABLE 1: MAUV RCS-3 TIMING

	State Clock Period (Executor cycle time)	Average rate of change of output	Planning Horizon
E-move	600 millisec	~10 seconds	~2 minutes
Vehicle Task	600 millisec	~1 minute	~10 minutes
Group	600 millisec	~5 minutes	~50 minutes
Mission	600 millisec	~30 minutes	~2 hours

7. World Modeling - M modules (Remember, Estimate, Predict, Evaluate)

The world model is the system's best estimate and evaluation of the history, current state, and possible future states of the world, including the states of the system being controlled. The world model includes both the M modules and a knowledge base stored in global memory. The world model thus corresponds to what is widely known in the literature as a blackboard [18].

The knowledge stored in the world model consists of state variables, maps, lists of objects, tasks, and events, and attributes of objects, tasks, and events. The world model includes both a priori information which may be provided to the system before a mission begins, and a posterior knowledge which is gained from sensing the environment as the mission proceeds.

As shown in Figure 8, the M modules at each level perform the following functions:

a) Maintain the global memory knowledge base, keeping it current. The M modules update the knowledge base based on correlations and differences between model predictions and sensory observations. This is illustrated in Figure 32.

b) Provide predictions of expected sensory input to the corresponding G modules, based on the state of the task and estimates of the external world, as shown in Figure 32.

c) Answer "What is?" questions asked by the planners and executors in the corresponding level H modules. The task executor requests information about the state of the world, and uses the answers to monitor and servo the task, and/or to branch on conditions to subtasks that accomplish the task goal. See Figure 33.

d) Answer "What if?" questions asked by the planners in the corresponding level H modules. As shown in Figure 19, the M modules predict the results of hypothesized actions.

e) The M modules also contain a set of values, and a process which evaluates the current situation and potential future consequences of hypothesized actions by applying evaluation functions to current states and to future states expected to result from hypothesized actions. The evaluation functions have as variables the set of values assigned to events such as vehicle survival, subtask completion, and information gathered by the vehicles. They also have as coefficients of those variables, the set of priorities assigned to each of the values. Values such as risk and payoff may be assigned to regions on maps. Cost and risk values may also be associated with map route segments.

Mission objective priorities are defined, and values are assigned to vehicles, targets, and resources at the beginning of the mission. These are typically not changed during the mission. Lower level task priorities are derived from the mission level priorities in the context of specific situations and state variables contained in the world model.

The evaluation functions use the priorities and values to provide value driven logic [30] for planning and execution at several hierarchical levels. The planners use the world model predictors and evaluation functions to search the space of possible futures, and choose the sequence of planned actions that produce the best evaluation. The executors are also able to apply value driven logic to the current state of the world in order to produce moment by moment behavioral decisions.

7.1 Global Memory

Global memory is the database wherein is stored knowledge about the state of the world including the internal state of the control system.

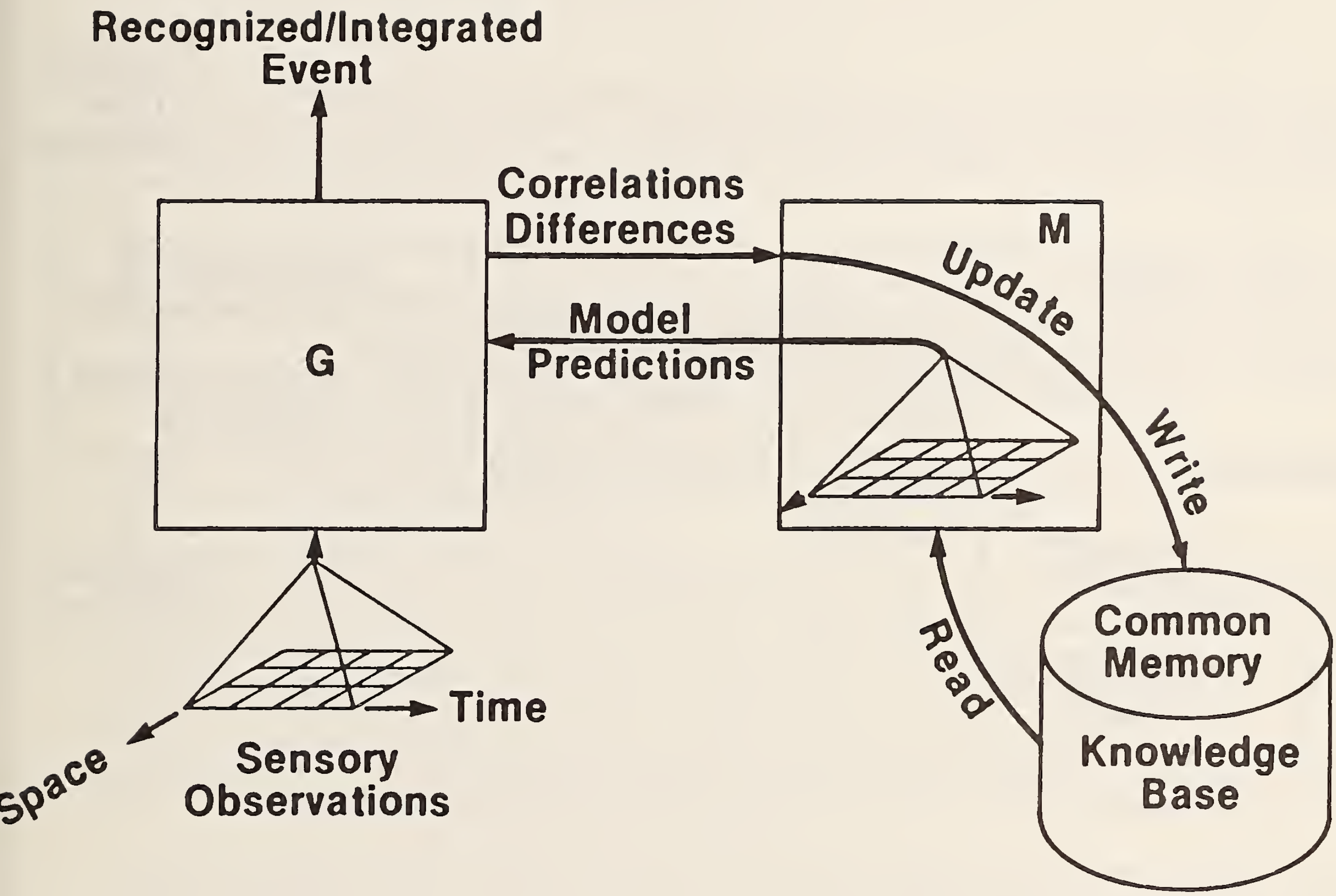


FIGURE 32. Role of M module in predicting sensory input and in up-dating knowledge base based in correlations and differences between predictions and observations.

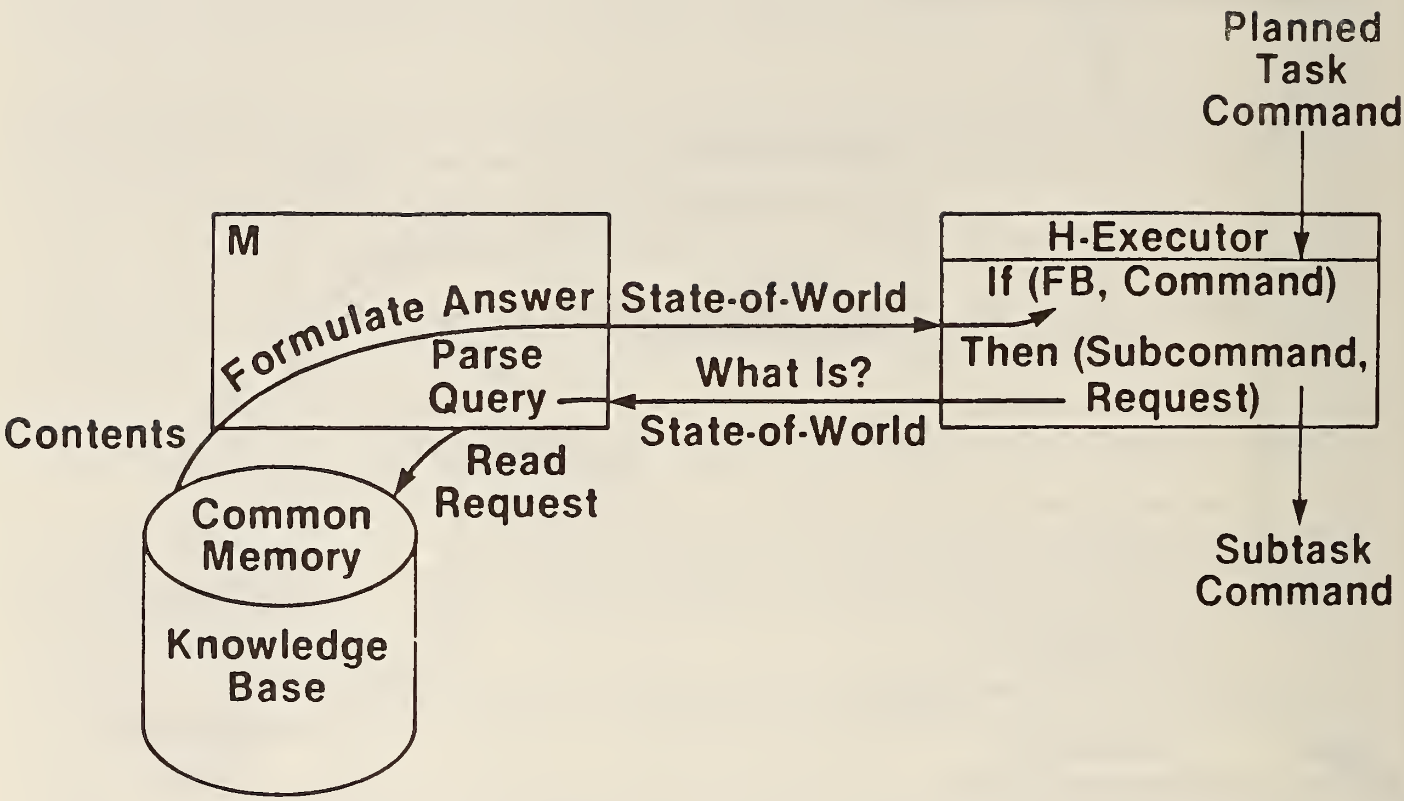


FIGURE 33. Role of M modules in responding to H module executor "What is?" questions.

7.1.1 Contents of Global Memory

The knowledge in the global memory consists of:

a) Maps

Maps describe the spatial occupancy of the world. A map is a spatially indexed database showing the relative position of objects and regions. Maps may also contain overlays, which may indicate values such as utility, cost, risk, etc. assigned to regions or objects on the map. These values can be used for planning and execution of tasks.

There are two types of map coordinate frames of importance to the MAUV project: world coordinates, and vehicle coordinates. These are illustrated in Figure 34. A world coordinate map is a two dimensional representation in which latitude and longitude are the x-y coordinates, and each pixel contains a pointer to a data structure that gives the physical properties and z-dimension of the region or objects covered by that pixel. Objects with vertical dimensions are projected onto the x-y plane of the map, and regions of constant height (or depth) may be indicated by contour lines.

A vehicle moving through the world can be represented as an object moving on the world map. The world map may be scrolled so as to keep a particular vehicle of interest at the center.

A vehicle coordinate representation of a map is also shown in Figure 34. The vehicle coordinate map is a polar coordinate system centered on the vehicle. Pixels are referenced by range and bearing. The vehicle coordinate map is derived from the world coordinate map. The contents of the pixels in the vehicle map change as the vehicle moves. This implies that the vehicle coordinate map must be periodically recomputed from the world coordinate map at a rate such that significant errors do not occur in the position of important objects on the map. In some cases, it may be convenient to have the vehicle coordinates represented on a log polar plot, where the range to pixels is represented on a logarithmic scale. This provides high resolution for near objects, and low resolution for distant. Objects at infinity may then be arranged around the outer edge of the vehicle centered world map.

The MAUV world model has a global database in which the world map is stored in quadtree form. This is illustrated in Figure 35 [43]. The minimum resolution of the quadtree is one half meter. The quadtree is an efficient structure for storage, but not for updating or scrolling. Therefore, the portion of the world map that is relevant to tasks being performed at the various hierarchical levels are transformed from the quadtree into a hierarchy of local world maps, each of which has the form of a 256x256 pixel array.

For each different hierarchical level, the local array map has a different resolution. Local array map resolution increases at each successively lower level, while the area covered by the local array map increases at each successively higher level. At each level, the local array map typically covers a region which completely contains the task being planned at that level. It has a resolution which is sufficiently fine grained so that subtasks being planned at that level are easily resolved. Local array maps at different levels thus represent a pyramid structure as shown in Figure 36.

These local pixel arrays are initialized so as to be approximately centered on the vehicle or group performing the task. As the vehicle or group moves away from the center of a local array map, the updated map is transformed back into the quadtree, and a new portion of the global world map, with the vehicle again at the center, is transformed into the local array map. As the vehicle moves through the world, the local array maps thus form a series of overlapping windows on the global world quadtree map.

Multiple Autonomous Undersea Vehicles
NBS DARPA

WORLD MODEL
MAPS

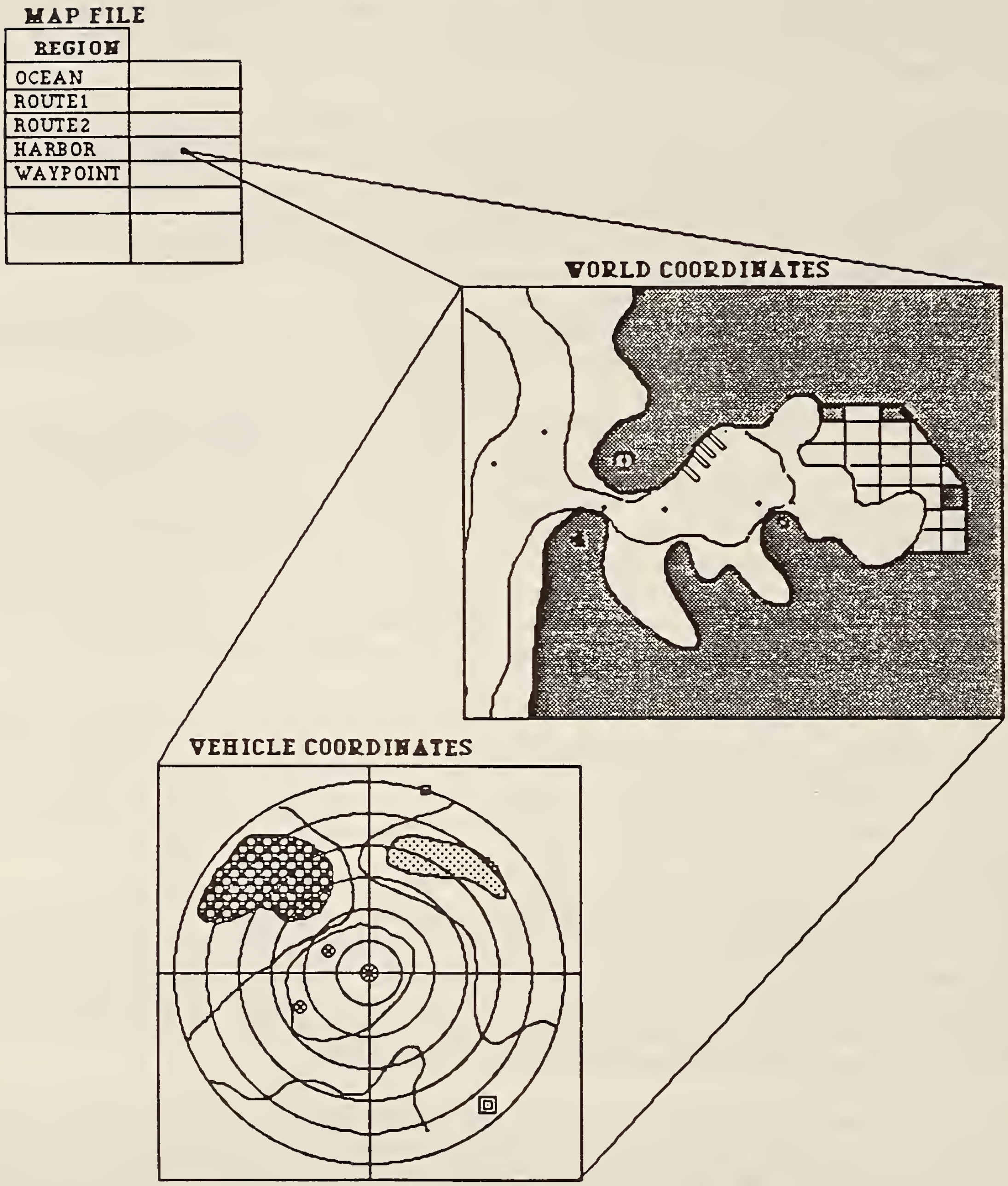


FIGURE 34. **World Model map representations.**

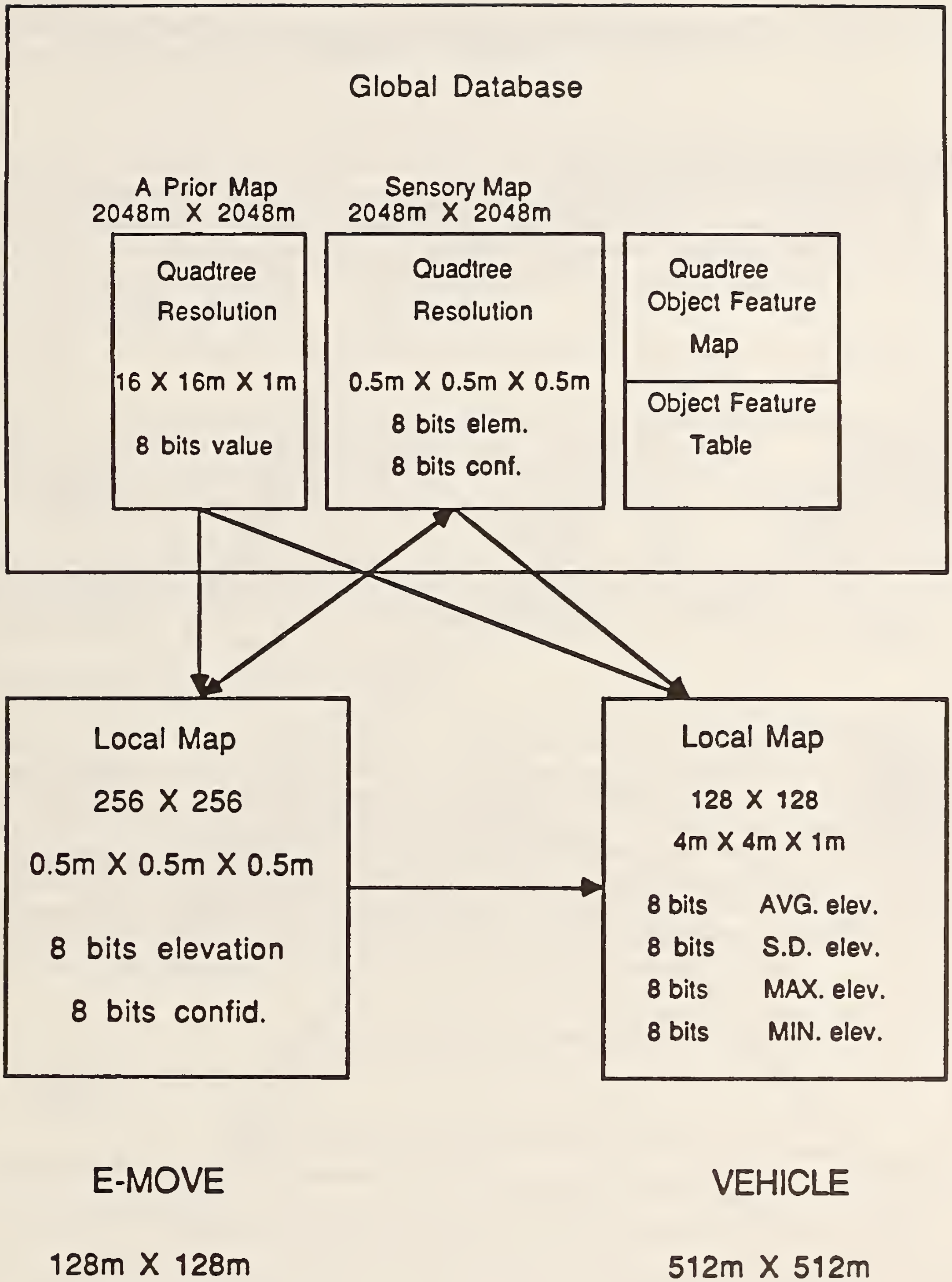


FIGURE 35. Global database storage of maps.

WORLD MODEL MAPS

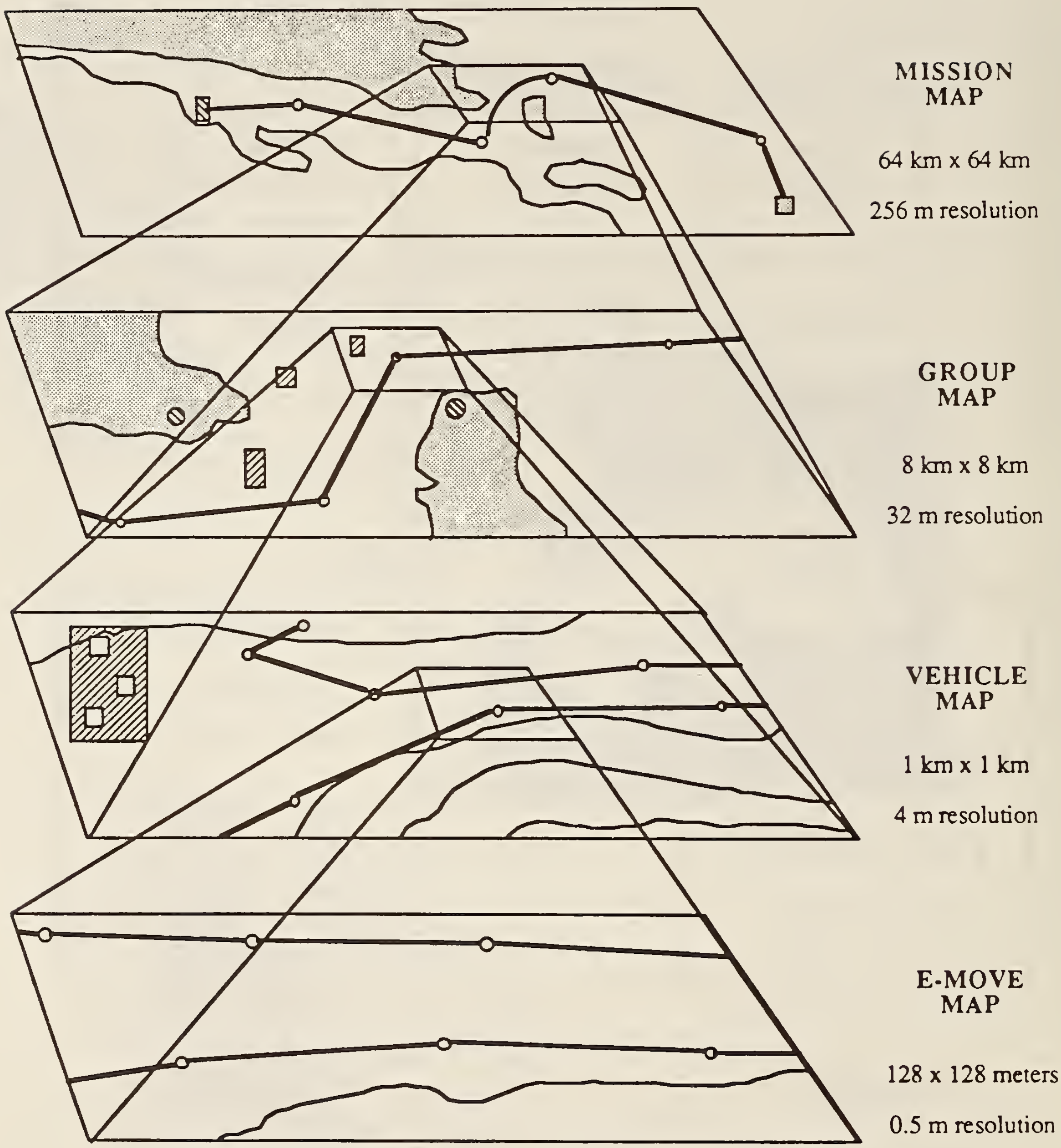


FIGURE 36. **Different resolutions of local maps for different hierarchical levels.**

These local world map arrays can then be further transformed into vehicle centered maps, or egospheres, and updated with sensory data.

b) Lists

All known objects, tasks, features, regions, and relationships, and events are listed in the global memory database indexed by name, and characteristic features. Each item in the list has a data form, or "frame", containing its attributes, as shown in Figure 37. Object frames contain information such as position, velocity, orientation, shape, dimensions, reflectance, color, mass, and other information of interest. For moving objects, the object frames contain not only current map coordinates, but a past history or trace of coordinate positions.

c) State Variables

The state variables in global memory are the system's best estimate of the state of the world, including both the external environment and the internal state of the H, M, and G modules.
Events are state vectors which include the time variable. Event vectors or event frames contain information such as start and end time, duration, type, cost, payoff, etc.

Recognized objects and events may also have associated with them confidence levels, and degrees of believability and dimensional uncertainty. At different hierarchical levels, object frames have different levels of detail and spatial resolution, and event frames have different levels of temporal resolution.

7.1.2 Implementation of Common Memory

Common memory in the MAUV architecture is not located in a single physical database, but is distributed over several computers, memory boards, and mass storage devices on a VME bus. Common memory is, in fact, distributed over more than one vehicle. Variables in common memory are globally defined, i.e., they may be accessed (read or written) by name from local processes running at any level. Of course, the time required to access a global variable is not the same for all processes. For example, in order for a global variable in vehicle-A to be read or updated by a process in vehicle-B, the two vehicles may have to rendezvous and communicate world model updates. This may take many minutes or hours. In the mean time vehicle-B would be forced to use its own local copy of the global variables, with the knowledge that it is not current, and therefore possibly incorrect.

WORLD MODEL

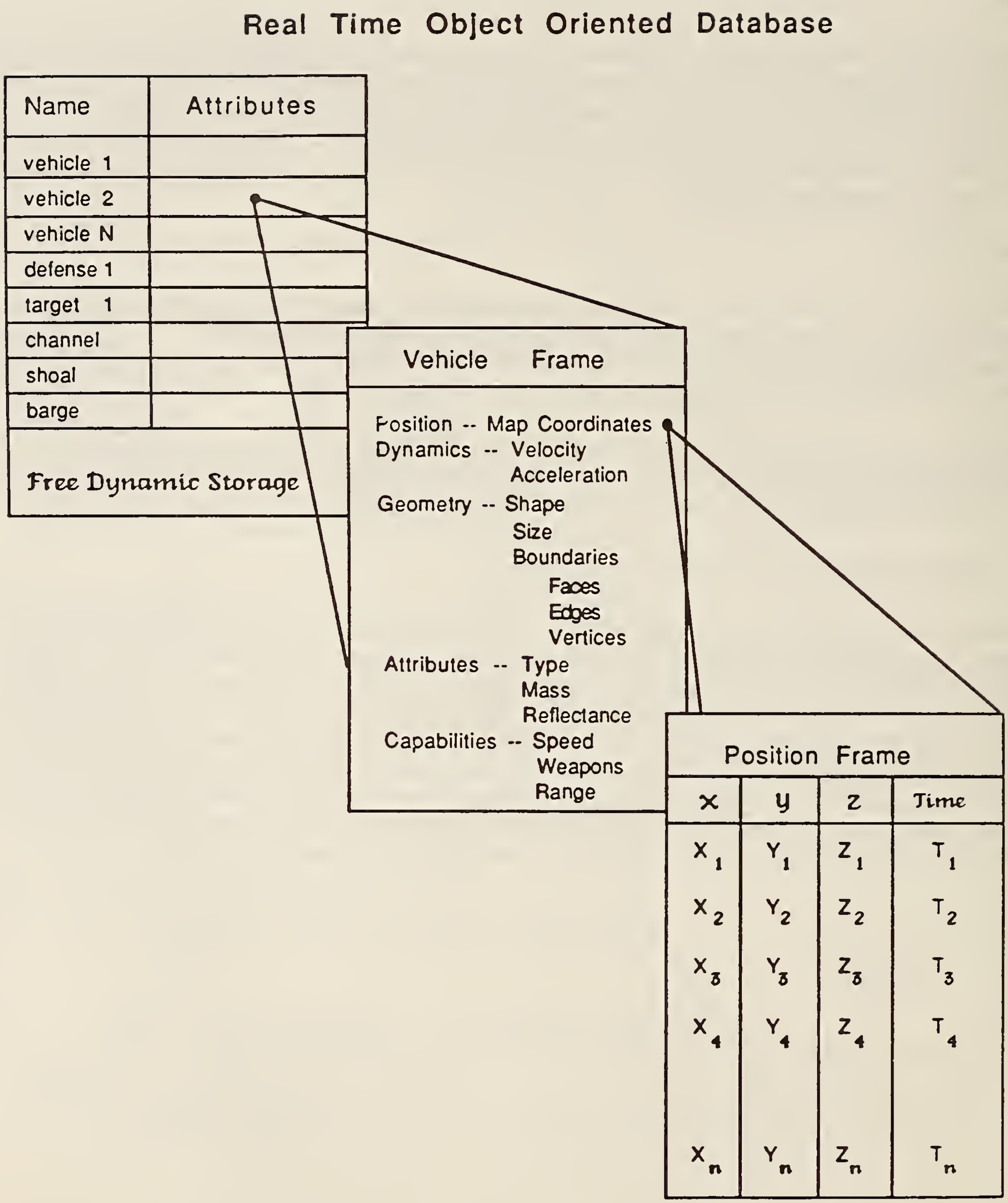


FIGURE 37. Object database in world model.

8. Sensory Processing - G modules (Filter, Integrate, Detect, Measure)

The sensory processing leg of the MAUV control hierarchy consists of G modules which recognize patterns, detect events, and filter and integrate sensory information over space and time. As shown in Figure 7, the G modules are dual to the H modules. They also consist of three sublevels which:

1) compare sensor observations with world model predictions

2) integrate correlation and difference over time

3) integrate correlation and difference over space

These spatial and temporal integrations fuse sensory information from multiple sources over extended time intervals.

Newly detected or recognized events, objects, and relationships are entered by the M modules into the world model global memory database, and objects or relationships perceived to no longer exist are removed. The G modules also contain functions which can compute confidence factors and probabilities of recognized events, and statistical estimates of stochastic state variable values.

8.1 Egospheres

An egosphere is a two dimensional representation of the world projected onto the surface of a sphere [44, 45]. It is obtained by placing a transparent sphere of unit radius around a vehicle (or group), and projecting the world onto that sphere. The relationship between the world map and an egosphere is shown in Figure 38.

Pixels on the egosphere contain pointers to data structures that indicate range and surface properties (such as reflectance) of the region or objects covered by that pixel. Regions of constant range can be indicated by contour lines. Objects are projected onto the surface of the egosphere where the line of sight from the origin to the object intersects the egosphere. The egosphere is thus a view of the world as seen from an individual vehicle (or from the center of a group).

There are a number of different egosphere representations:

1) A sensor (camera) egosphere is shown in Figure 39. The rows and columns in the image of the camera field of view of the camera can be represented either by z and x coordinates, or by azimuth and elevation. For narrow field of view, these representations are essentially the same. For wide field of view, azimuth and elevation are preferrable because there are fewer problems with distortion and edge effects

2) A vehicle egosphere is shown in Figure 40. It has coordinates of azimuth and elevation measured in a reference frame fixed in the vehicle chassis.

3) An inertial egosphere is shown in Figure 41. It has coordinates of azimuth measured east from north, and elevation measured up from the horizon. It has the advantage that distant objects remain fixed despite vehicle rotation or small amounts of translation.

4) A velocity egosphere, is shown in Figure 42. The velocity vector defines the positive z-axis (or pole), and gravity defines the plane of zero azimuth. The velocity egosphere representation is well suited for dealing with image flow. As the vehicle moves, the positive z-axis corresponds to the focus of expansion. For stationary objects in the environment, image pixels flow along great circle arcs of constant azimuth.

WORLD MAP/EGOSPHERE TRANSFORMATION

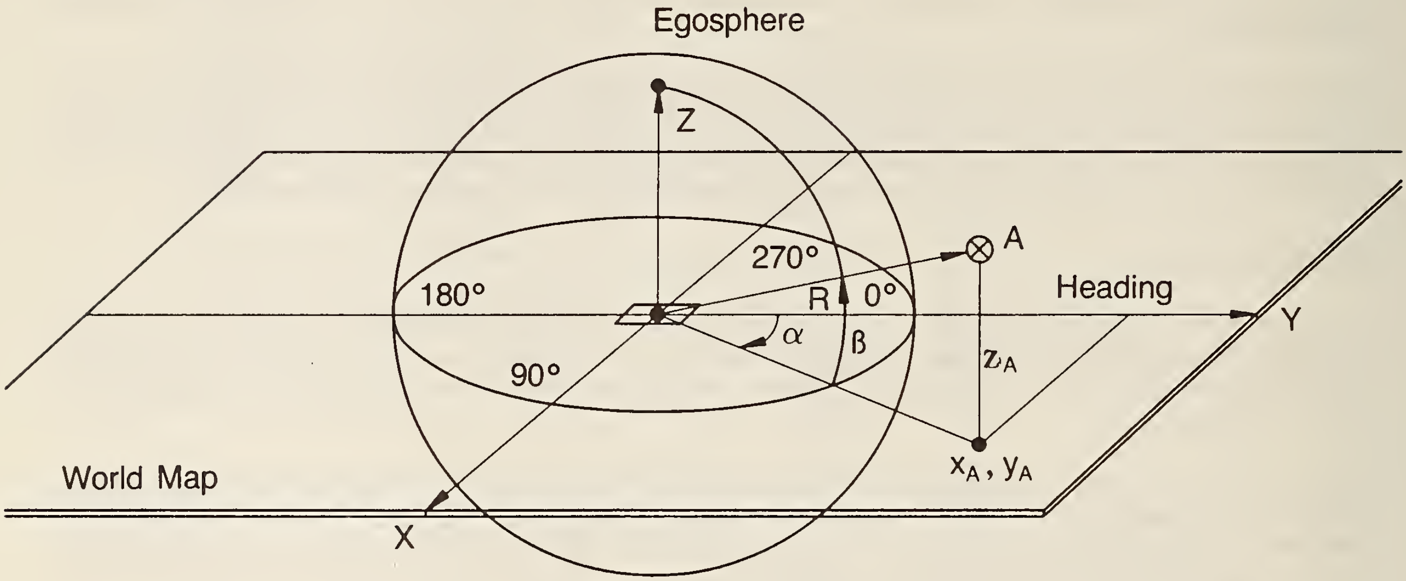


FIGURE 38. **Geometric relationship between world map and egosphere.**

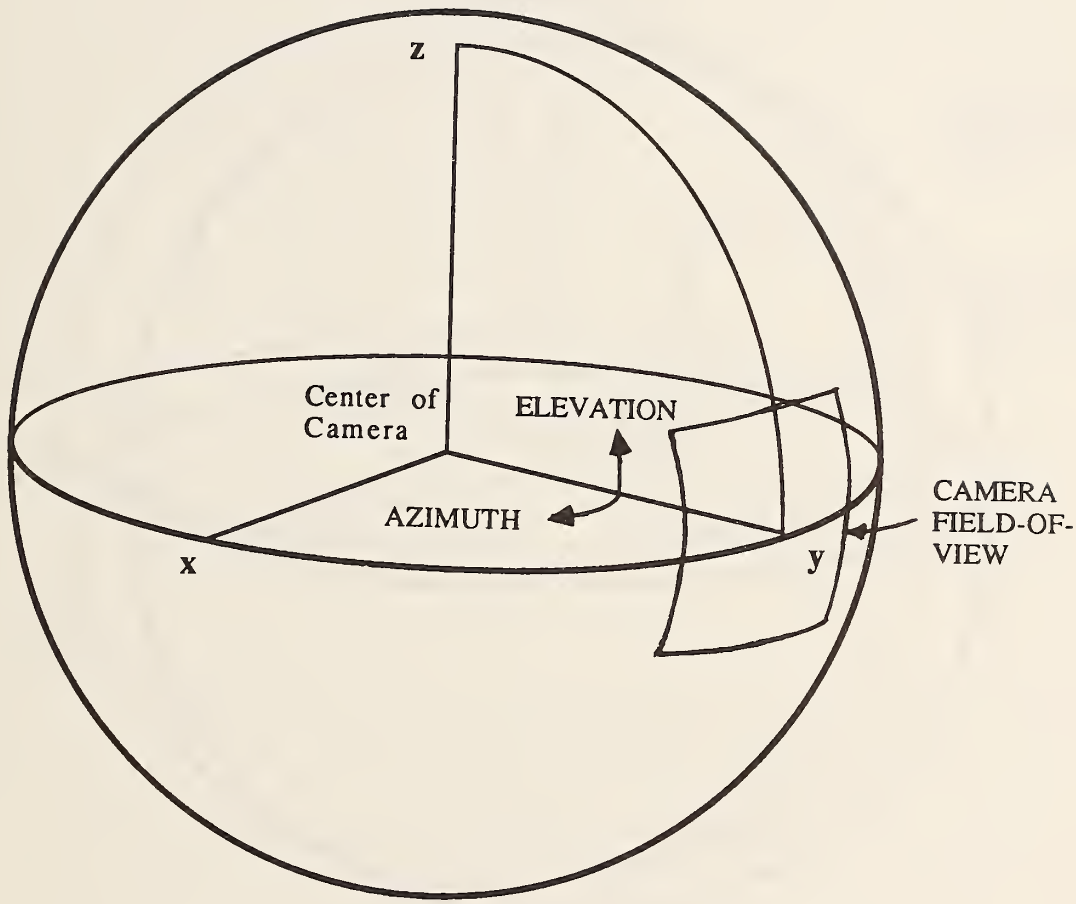


FIGURE 39. **Camera egosphere.**

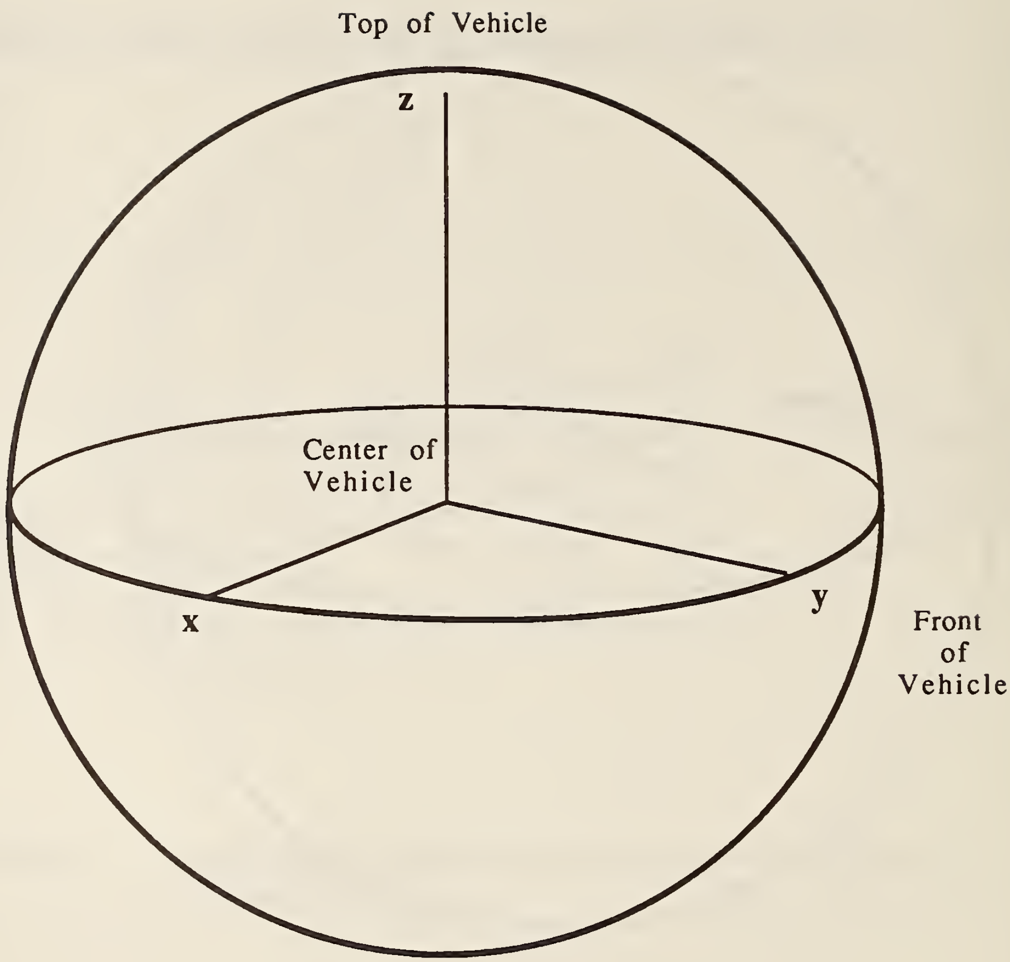


FIGURE 40. **Vehicle egosphere.**

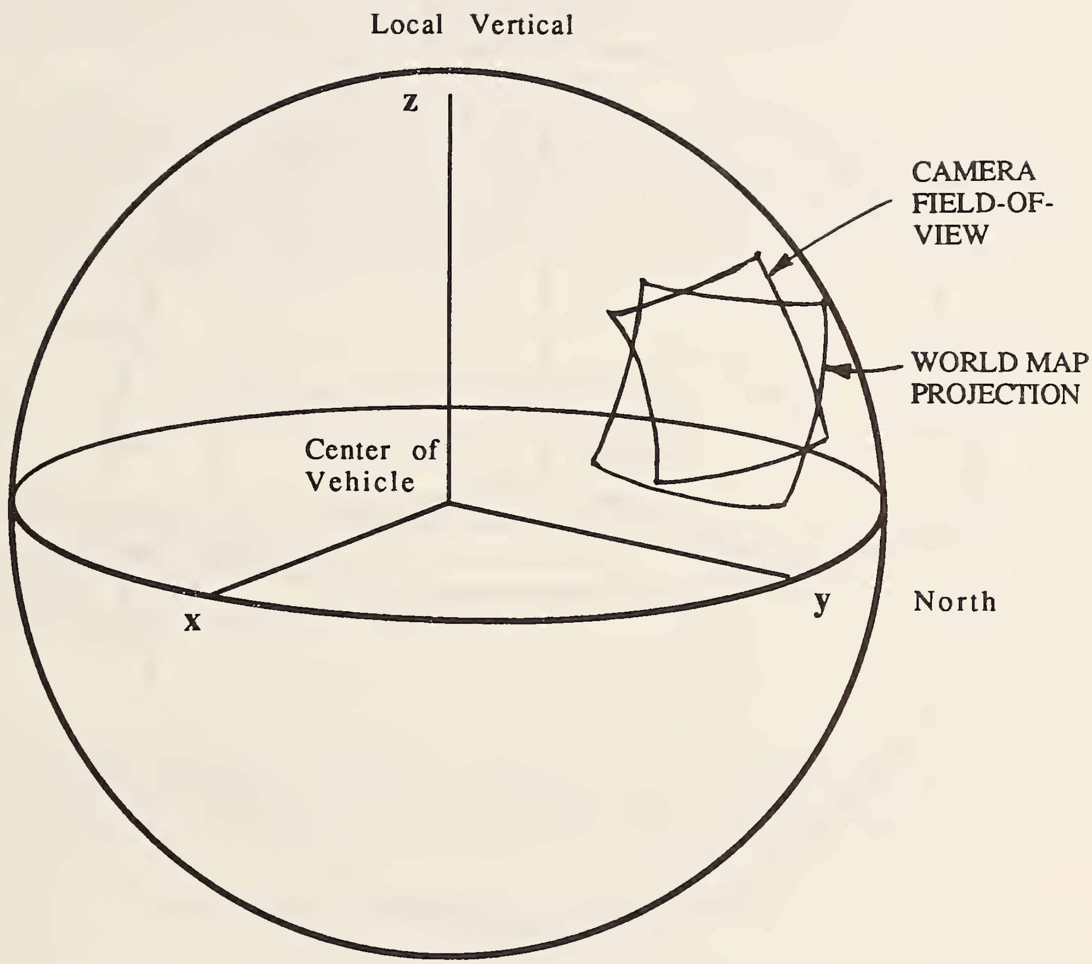


FIGURE 41. **Inertial egosphere.**

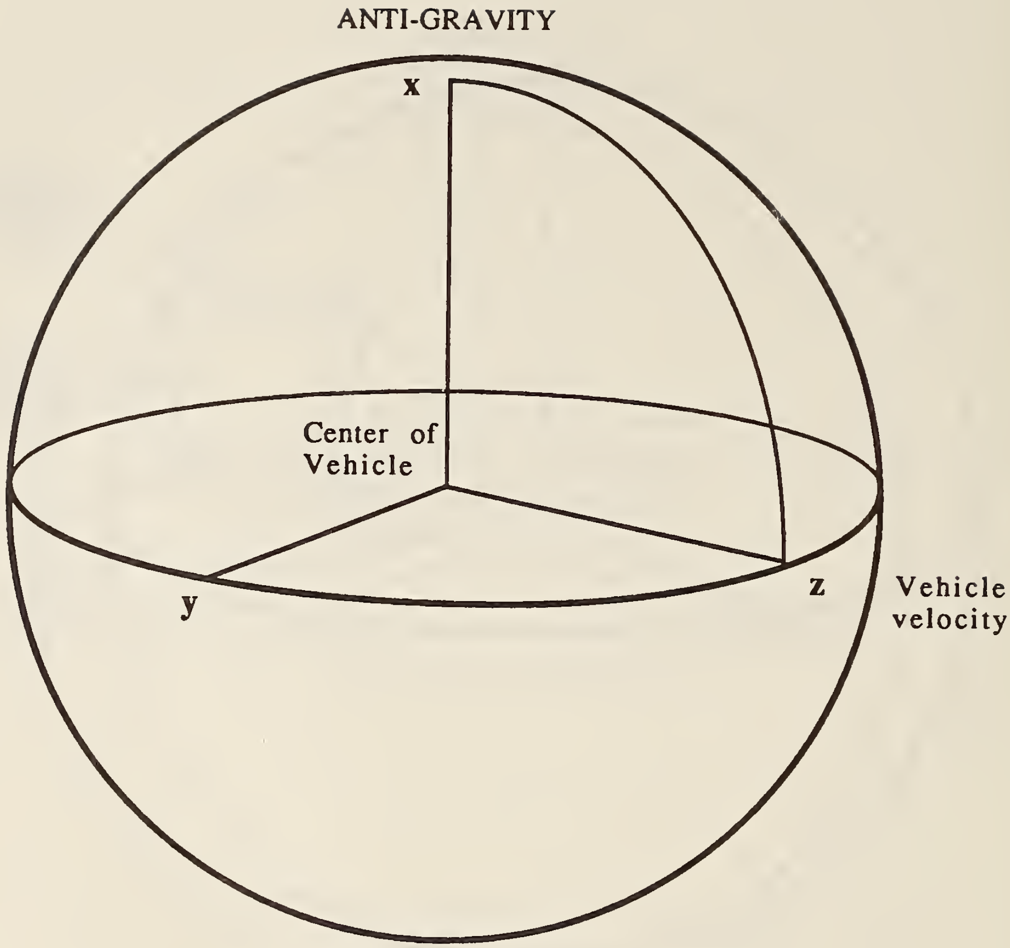


FIGURE 42. Velocity egosphere.

The vehicle egosphere representation is well suited for fusing sensory data from multiple sensors. For example, range data from a sonar sensor can be overlaid on the vehicle egosphere with vision data from a camera. Range data from multiple sonar sensors (or if the vehicle is stationary, multiple readings of the same sensor) can be overlaid on the vehicle egosphere to build up an image.

The inertial egosphere is well suited for fusing multiple sensor readings over time on a moving or rotating vehicle. The inertial egosphere is also ideal for comparing sensory input with world model predictions from stored map data. If sensory data is overlaid with world map data on the egosphere, each brightness pixel from the camera will be overlaid with range data from the world map. Conversely, objects observed in the image will be overlaid on objects predicted in the map.

The velocity egosphere is ideal for computing image flow due to motion of the vehicle through the world. As shown in Figure 43, objects in the world appear to radiate outward from the positive z-axis, and converge to a point at the negative z-axis as the vehicle moves through the world. The velocity of image flow for each point on the velocity egosphere is a simple function of velocity, range, and elevation angle on the egosphere. For stationary objects in the world, the image flow equations for egosphere pixels are given by equations (1) and (2)

(1) $dA/dt = v(\sin A)/r$

(2) $dB/dt = 0$

where A is the angle between the camera velocity vector and the egosphere pixel
r is the range to the object covered by the pixel

and v is the velocity of the camera

Vehicle velocity is typically known. Predicted image flow requires range data for each pixel. Predicted range can be obtained from the world map. Observed image flow can be used to compute range for each pixel. If world map data is overlaid on the velocity egosphere, differences between observed and predicted image flow can be used to correct object positions predicted from the map.

As shown in Figure 44, the image flow equations for moving objects are given by equations (3) and (4) as

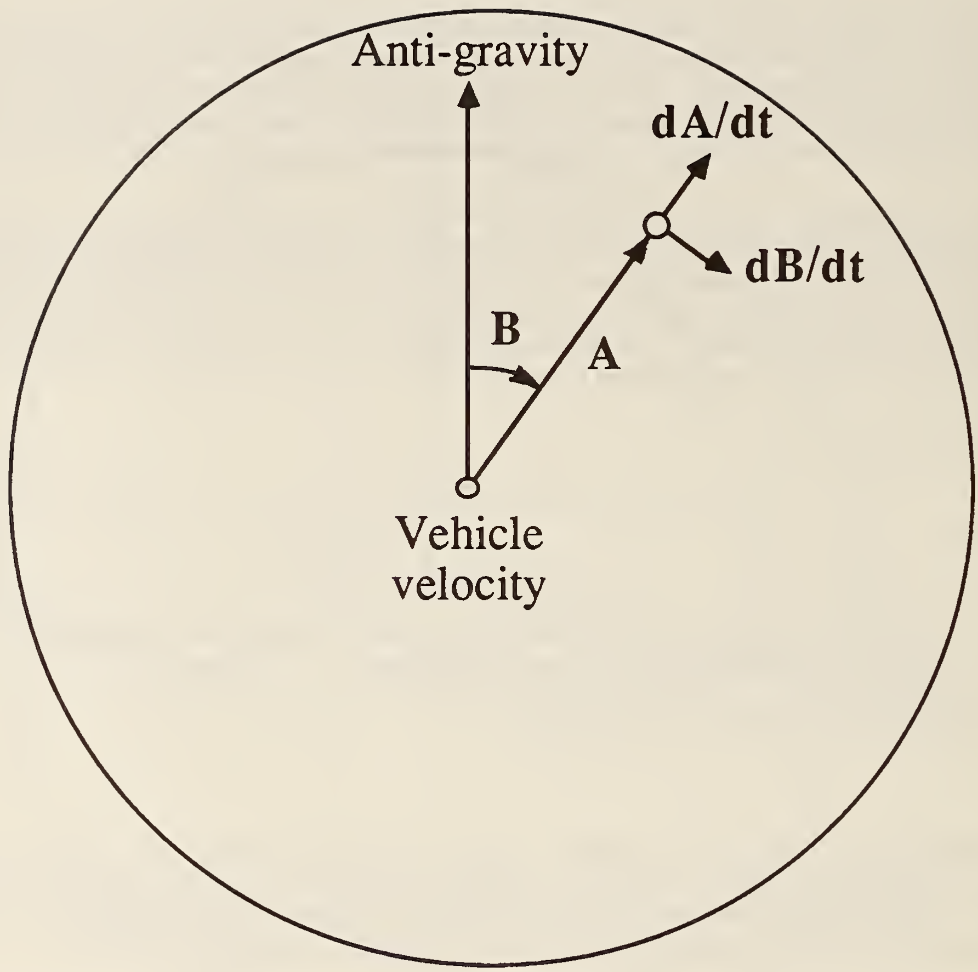
(3) $dA/dt = (v - v_z)(\sin A)/r + v_y (\cos A)/r$

(4) $dB/dt = -v_x / (r \sin A)$

where v_z is the object velocity along the vehicle velocity vector

v_y is the object velocity perpendicular away from the vehicle velocity vector
and v_x is the object velocity perpendicular to v_y and v_z

Figure 45 shows the transformations between the various egosphere representation. Landmark

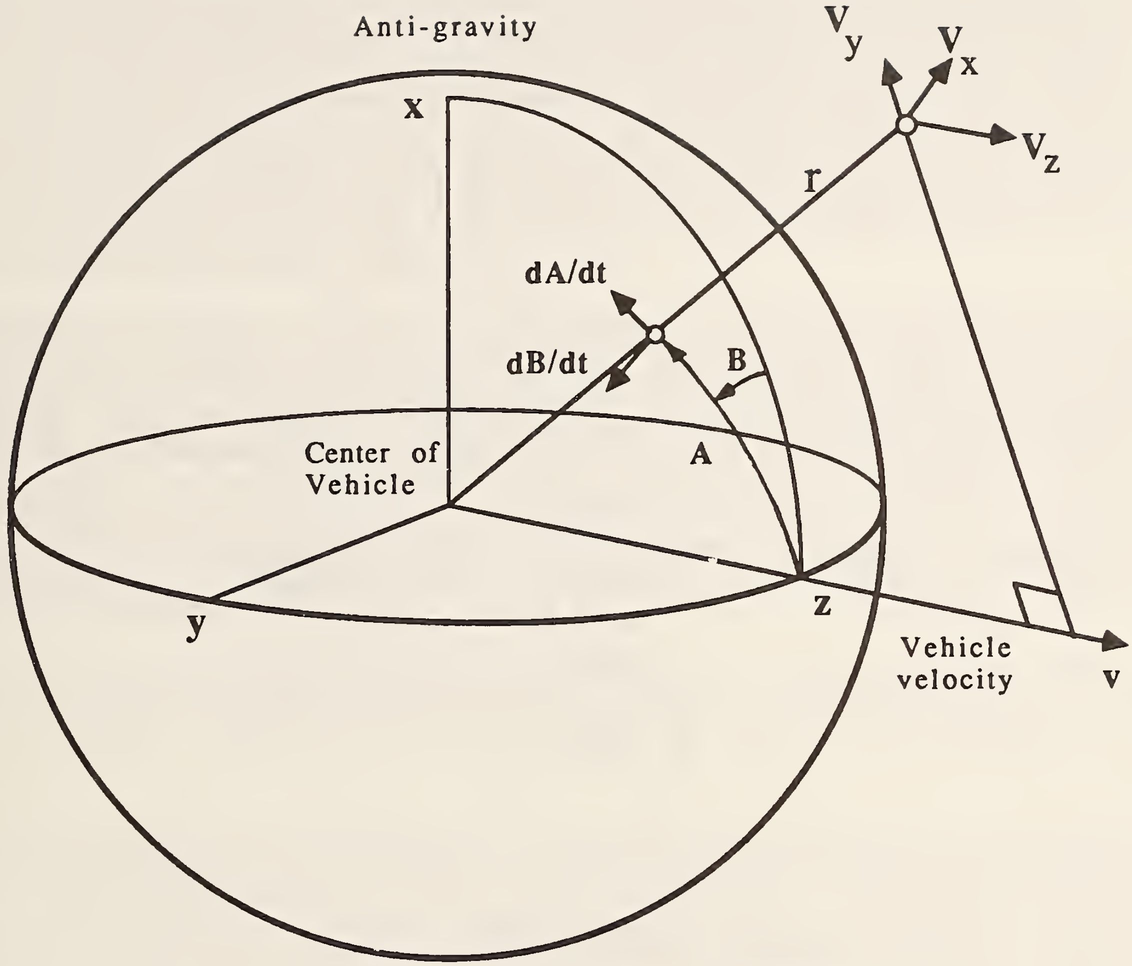


For stationary objects:

$$dA/dt = (\sin A / r)\, v$$

$$dB/dt = 0$$

FIGURE 43. View from center of velocity egosphere



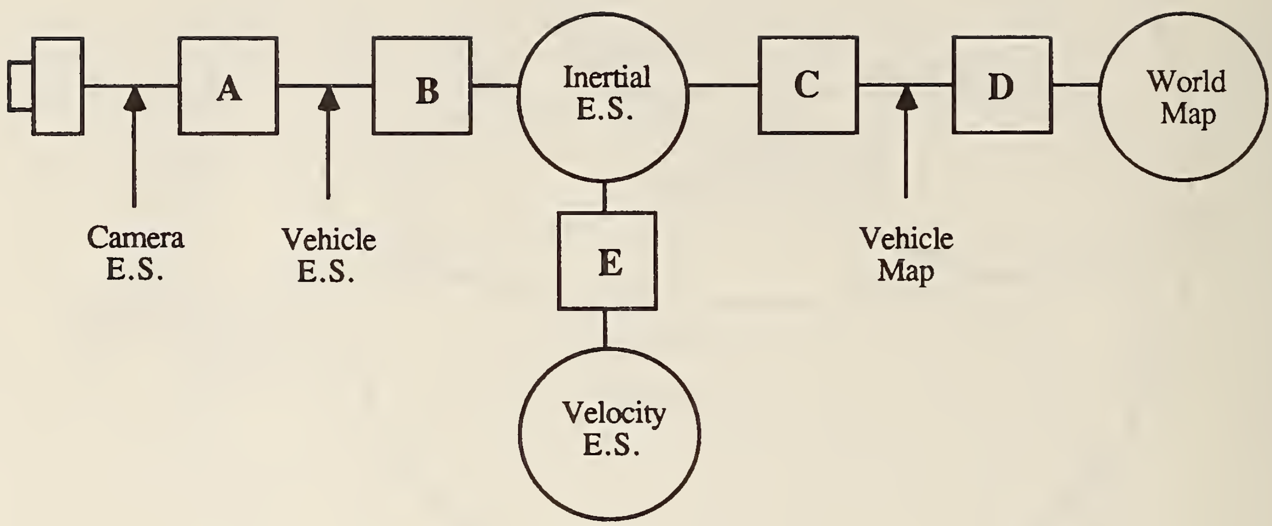
For Moving Objects:

v_z parallel to v
v_y perpendicular away from v

$$dA/dt = (\cos A / r)\, v_y + (\sin A / r)\, (v - v_z)$$

$$dB/dt = -v_x / (r \sin A)$$

FIGURE 44. Velocity egosphere.



A = f (camera pan, tilt, roll)

B = f (vehicle roll, pitch, yaw)

C = f (vehicle z)

D = f (vehicle x, y, heading)

E = f (camera velocity vector)

FIGURE 45. Transformations for matching camera data to world map data.

navigation can be accomplished by matching sensor data with world map data on the egosphere. When sensor data correlates with map data on the egosphere, the vehicle position on the map is correct. When sensor data does not correlate with map data, the direction and approximate magnitude of the error in vehicle map position can be determined by computing on the egosphere the approximate displacement of the vehicle needed to produce the image flow required to null the difference between sensor observations and map predictions. The position of the vehicle on the map can then be corrected (or the vehicle can be physically driven to a new location), and another comparison between sensor and map data on the egosphere can be made. This error correction process will converge rapidly so as to "servo" the vehicle position into the correct map position.

Vehicle maneuvers relative to objects on the egosphere can often be computed directly from simple trigonometry on the egosphere data. For example, steering around an obstacle can be accomplished by finding the point on the edge of the obstacle that lies closest to the vehicle motion vector (i.e. has the smallest value of A) on the egosphere, and steering in that direction. Another example, if an incoming missile on the egosphere exhibits motion, it is probably not an immediate threat, since an object which displays motion on the egosphere is not moving toward the center of the egosphere. A reasonable avoidance strategy is to steer in a direction opposite from the apparent motion. If, however, the missile appears motionless and growing larger in size, then it is an immediate threat, for it is coming directly at the egosphere center. An avoidance strategy is to immediately steer anywhere in a plane 90 degrees from the missile's image.

Both egospheres and world maps can have overlays which indicate values to be used for planning and execution. For example, a region on a map or on an egosphere may be labeled as enemy territory, and assigned a risk value. If stealth has a priority, motion can be planned so as to limit exposure to that region.

Both egospheres and local world map arrays have varying resolution depending on hierarchical level. For egospheres, the relation between resolution and hierarchical level is not necessarily the same as that of the local world maps. Resolution on the egosphere is determined by the resolution of the sensor systems using the egospheres. In the case of air or land vehicles using optical sensors, the resolution of the sensor system increases with range, and hence with higher levels in the hierarchy. As illustrated in Figure 46, high resolution narrow field-of-view sensors are typically used for long range measurements, while low resolution wide-field-of-view sensors are typically used for short range measurements. Long range sensing is typically relevant to high level task decomposition, while short range sensing is relevant to low level task decomposition.

For optical sensors in air, the percentage of the sphere accessed by the sensor system at any instant of time decreases at each successively higher level. However, for underwater sonar systems, resolution rarely exceeds 1 degree. Long range sonar typically has much lower resolution. Thus, for the MAUV project, only a low resolution (approximately one degree per pixel) egosphere representation will be developed. The MAUV egosphere will cover the entire egosphere with 55,024 resolution elements. This representation will be used for obstacle avoidance and to fuse data from all types of acoustic sensors.

If a vision system were added to the MAUV vehicle, a second egosphere representation would be added. This would have a resolution in which the field of view of the camera (about 45x45 degrees) would contain 256x256 resolution elements. The camera field of view would thus create a 45x45 degree patch on the surface of the egosphere which would be moved over the egosphere to match movements in the camera pointing system. Input from the camera would thus "paint" video image data onto the egosphere. As the camera pans and tilts, it would leave a trail of data, which would grow old with time. The goal of the camera pointing system would be to keep the camera pointed at regions on the egosphere where action is occurring, or to regions where objects relevant to the task are located. The camera pointing system would thus allocate dwell time, or "attention", of the vision system.

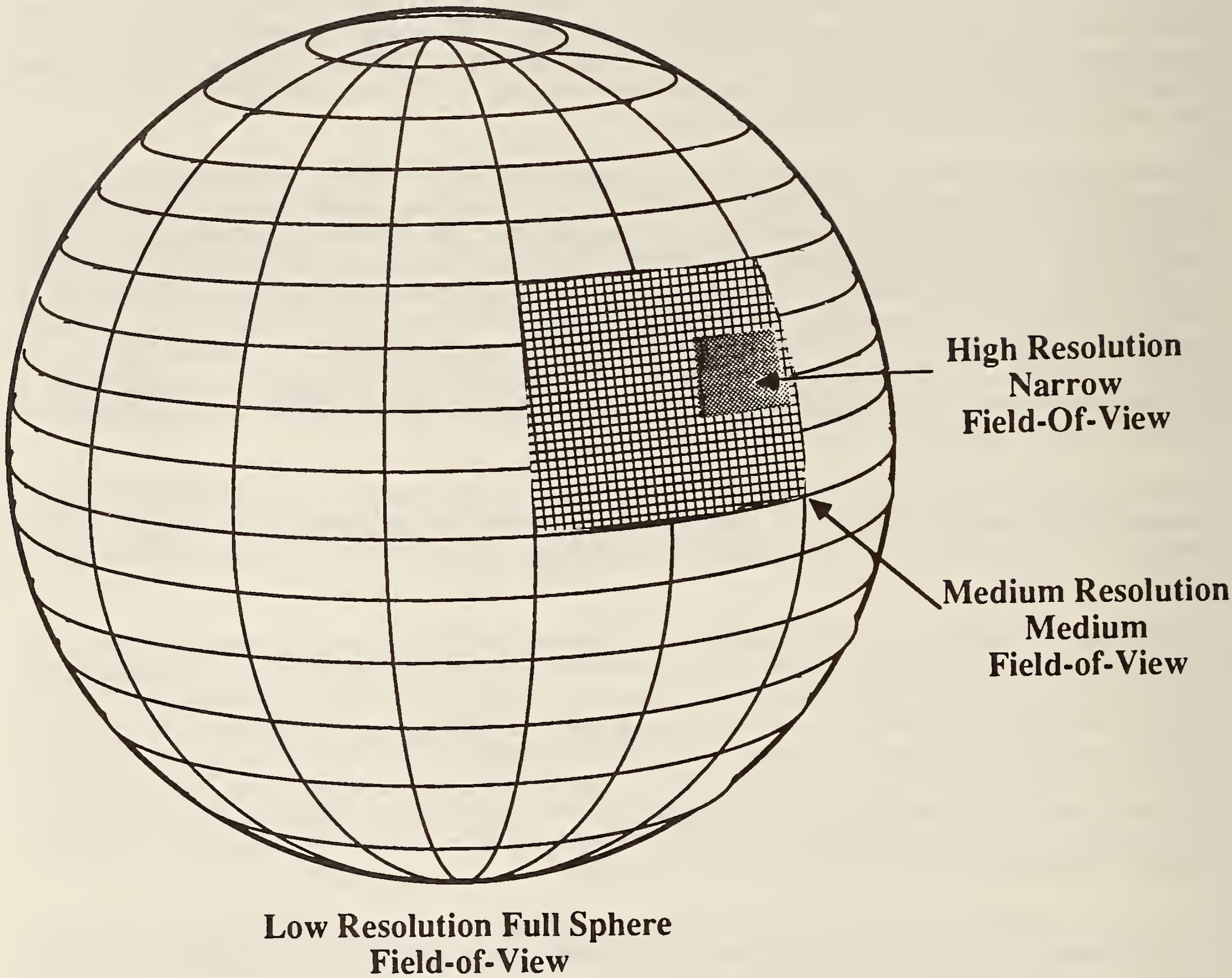


FIGURE 46. Egosphere for camera system with three levels of resolution. Each field of view consists of 256 x 256 pixel array.

On a land or air vehicle, this second egosphere representation could be used for vehicle level navigation. For manipulation, this egosphere representation would be used by the object/task level. On the MAUV this representation could be used for optically guided docking maneuvers, or for optical inspection of objects.

For air and land vehicles, a high resolution egosphere representation could also be provided to cover a 3x3 degree patch on the egosphere with 256x256 resolution elements. This representation would be used for fusing data from vision data collected through telescope optics. Such data would typically be relevant to group level control decisions. The 3x3 degree patch would move over the egosphere to match movements in the telescope pointing system.

The properties of these mid and high resolution egosphere representations are analogous to a foveal-peripheral vision system. Mid resolution camera optics would collect data relevant to vehicle task objects, and high resolution telescopes would collect data relevant to longer term group tasks.

Beyond the group level, the range to objects of interest on the egosphere is on the order of tens of miles. On this scale, virtually all objects and regions of interest are compressed into a flat plane, and hence the egosphere compresses into a vehicle centered world coordinate map.

The egosphere (and the vehicle centered coordinate map) representations have the disadvantage that the positions of objects change constantly with vehicle motion. As the vehicle moves through the world, the projections of objects on the egosphere flow across its surface. However, for an inertial egosphere, objects at infinity remain motionless regardless of vehicle motion. Only nearby objects exhibit motion parallax.

Once the position of the vehicle is known on the world map, the transformation of each pixel from the world map to the egosphere map (or vice versa) requires only a 3 x 3 matrix multiplication. If the map region of interest can be localized to a 256 x 256 section of the map, the egosphere to world map (or vice versa) transformation can be accomplished in real-time (i.e. 16 times per second) by a hardware vector multiplier board of conventional design. Hidden surface removal is accomplished automatically by transforming the most distant map pixels first, and over-writing with closer pixels.

Since the egosphere updates from sensors at all resolution levels involve images containing no more than 256x256 pixels, the speed of transformation from egosphere to world map, and vice versa, is independent of resolution level. Thus, the computation load of coordinate transformation for three levels of resolution is no more than three times that for one level. Furthermore, since motion parallax is smaller for distant objects, the slew rates of the mid and high resolution sensors can be kept low, making the computation load much less than three times, and potentially only slightly more than one times, that required for the lowest level.

9. Implementation of RCS-3

The current MAUV implementation of the RCS-3 control hierarchy is shown in Figure 47. In this configuration, RCS-3 runs under the pSOS real-time multi-tasking operating system on a VME bus. It uses pRISM for multi-processor distributed systems. pSOS is written in C and uses Unix for a program development environment. The pSOS/pRISM system provides communication between modules of the RCS-3 architecture.

Communications within the RCS-3 control hierarchy uses a common memory, in which state variables are globally defined on a 32 bit (4 gigebyte) virtual address space. The physical memory addresses reside on the various single board computers, the common memory board, and the optical disk.

Each module in the sensory processing, world modeling, and task decomposition hierarchies read inputs from, and write outputs to, the virtual common memory. Thus each module needs only to know where in global memory its input variables are stored, and where in global memory it should write its output variables. The data structures in the global memory then define the interfaces between the G, M, and H modules.

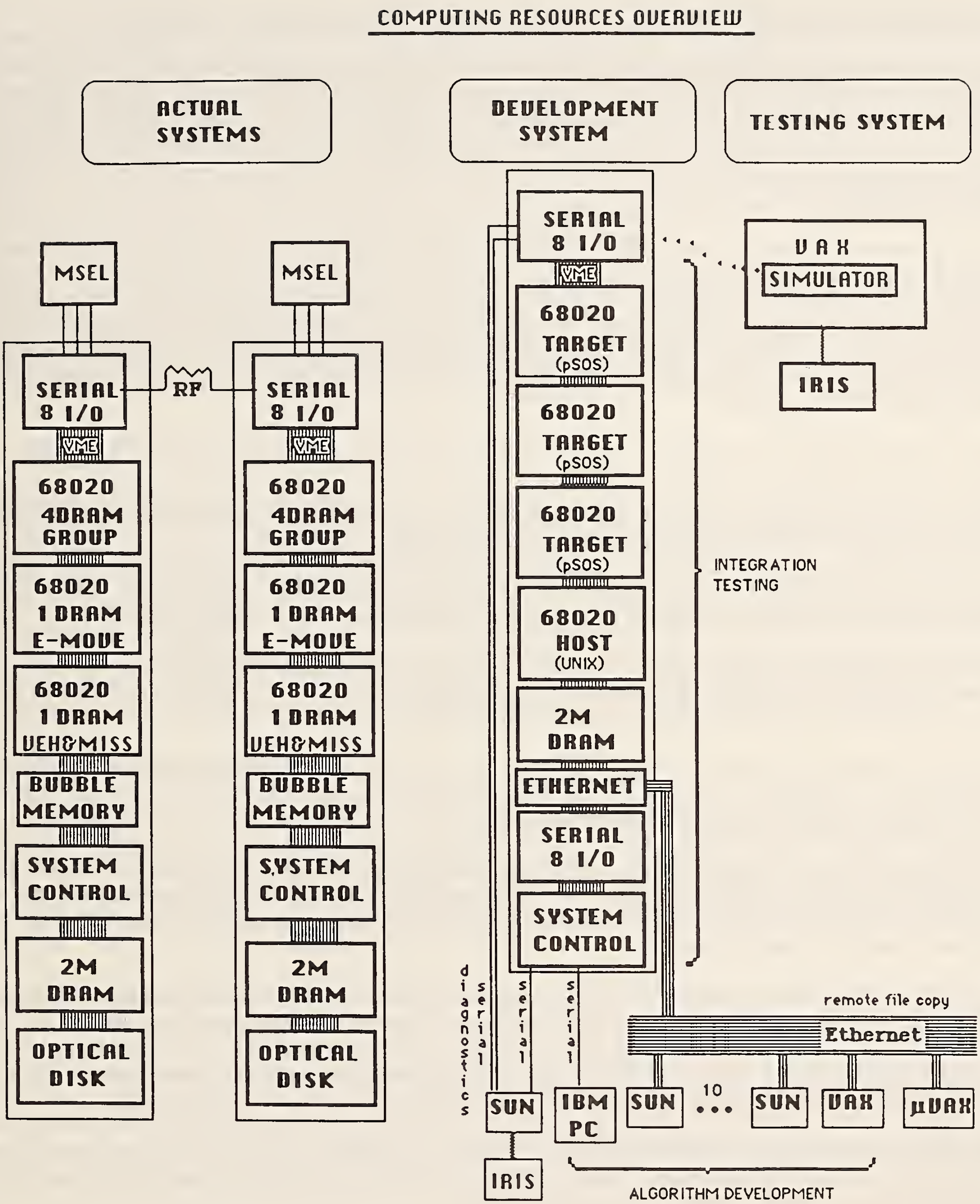


FIGURE 47: Implementation of RCS-3 on two MAUV vehicles plus the development system, simulator, and program development environment.

10. Operator Interfaces (Control, Observe, Define Goals, Indicate Objects)

The RCS-3 architecture is designed to serve both autonomous systems (such as MAUV) and telerobotic systems (such as the space station telerobot servicer). It thus has provisions for operator interfaces whereby a human can enter the control hierarchy at many different levels. When implemented, these interfaces can enable an operator to control low level functions, to define high level goals, to observe system operation at all levels, and to assist the sensory processing system if necessary by indicating features of objects.

For autonomous systems such as MAUV, most of the operator interfaces have not been implemented. However, even autonomous systems must be capable of responding to commands from a higher authority. Also, from time to time, there is the need for autonomous systems to respond to operator inputs for deployment and recovery, and for performing tasks that are beyond the capability of the autonomous system. Data interfaces are also needed for data logging, test, and debugging operations. The RCS-3 system for MAUV thus is designed with an operator interface to provide access to the control hierarchy at all levels.

10.1 Control Interface Levels

An operator control interface directly into the motor or actuator drivers (at the output of level 1) would permit the operator to control individual thruster drive currents.

An operator control interface into the task decomposition hierarchy at the input to the servos (at the middle of level 1) would permit the operator to control individual thruster rates, or forces.

A control interface at the input to level 1, would permit the operator to control a vehicle with a joy stick or steering wheel. The position, velocity, acceleration or force of the vehicle is controlled. In manipulation control, this is called resolved motion force/rate control.

A control interface at the input to level 2 (primitive level) would permit the operator to indicate motion way points on a local map. The vehicle would automatically compute acceleration profiles needed to produce dynamic motion through the indicated points.

A control interface at the input to level 3 (E-move) would permit the operator to describe key poses on an interactive graphics display system, or give symbolic commands for elemental movements (E-moves) such as <execute-S-turn>, <move-to-pose X>, <approach- target-feature Y>, etc. The input device may be menu driven.

A control interface at the input to level 4 (Vehicle) would permit the operator to designate objects, and define tasks to be done on those objects, such as <attack-target T>, <defuse-mine M>, etc.

A control interface at the input to level 5 (Group) would permit the operator to define group priorities, tactics, and assign group task elements to individual vehicles.

A control interface to level 6 (Mission) would permit the operator to change the mission entirely, or to alter strategy and mission priorities. This level can also select the operating mode of the system (i.e., <initialize>, <run>, <shut down>, <reconfigure>, etc.)

10.2 Monitoring Interfaces

Operator interfaces can permit a human to monitor the state of the system, including system state variables, world model, and sensor processing variables at any level. Windows into the common memory knowledge base permit viewing of maps, lists of recognized objects and events, object parameters, and state variables such as positions, velocities, forces, confidence levels, tolerances, traces of past history, plans for future actions, and current priorities and utility function values. Icons allow state variables to be displayed as dials, bar graphs, and time traces, or be represented as multiple exposures with time decay.

Sequences of past actions or plans for future action can be represented as state graphs, with windows into edges to display the conditions required for state transitions, and windows into nodes to display the state of the various modules in the control system at different times.

Geography and spatial occupancy can be displayed as a variety of maps, vectors, or stick figures. Object geometry can be represented with wire frame or 3-dimensional shaded graphics. The operator may also have a direct television image of the robot's sensory environment with graphics overlays which display the degree of correlation between the world model (i.e. what the robot believes is the state of the world) and what the human operator can observe via the sensory input with his own eyes.

10.3 Sensory Processing/World Modeling Interfaces

An operator interface may also permit human interaction with the sensory processing and/or world modeling modules. For example, an operator using a video monitor with a graphics overlay and a light pen or joystick can provide human interpretative assistance to the vision/world modeling system. The operator might interactively assist the model-matching algorithms by indicating with a light pen which features (e.g., edges, corners) in the image correspond to those in a stored model. Alternatively, an operator could use a joystick to line up a wireframe model with a TV image. The operator might either move the wireframe model so as to line it up with the image, or move the camera position so as to line up the image with the model. Once the alignment was nearly correct, the operator can allow the automatic matching algorithm to complete the match, and track future movements of the image.

10.4 Programming Interface

The operator interface can also be used as a programming tool. An expert system front end can be used to input and edit expert system rules. Each level of the RCS-3 control system can be represented as a state graph, where nodes are states, and edges are state transitions. The operator interface can permit the state graph to be edited by adding or deleting nodes and edges. It will also permit the nodes and edges of the state graph to be opened into windows containing the code that is represented by the nodes and edges. This code can then be edited on-line.

There also exists the prospect that neural net learning systems can be devised that will permit control functions to be learned by the neural net system observing the response of a human operator to a variety of control situations. This may permit system response (including tactics and strategies) to be taught by example. Using an operator interface to control the vehicle, the human operator may be able to teach the vehicle control system how to behave under a variety of circumstances by simply operating the system while the neural net system is in a learning mode. In this case, the operator need not be skilled in programming computers, but can teach the control system by simply showing it how a task should be done.

10.5 Operator Interface Mechanisms

The global representation of system state variables facilitates interaction with an operator, because it provides easy access for displaying variables and debugging the system. An operator interface process can be written to access system state variables in global memory by using the system dictionary. The interface process would provide the necessary translators to format human inputs into the proper format, and synchronize them with the control processes at the appropriate hierarchical levels. The concepts of a "time clutch" and "time brake" developed by Conway and Volz [46] are examples of how synchronization and hand-off can be accomplished.

By these mechanisms, it is possible for a human operator to enter the control hierarchy at any level, at virtually any time of his choosing, to monitor a process, to insert information, to interrupt automatic operation and take control of the task being performed. The operator may also apply human intelligence to sensory processing or world modeling functions.

11. Detailed Description of RCS-3 Control Levels for MAUV

11.1 Level 6 - Mission Control Level

Input commands to the H module at the mission control level will come from fleet battle management. They will consist of commands to one or more groups to perform a specific mission. The mission may be to respond to specific enemy threats or to defend against or attack specific targets or regions. For example, mission control will be capable of executing the following type of commands:

Mine harbor X
Clear mines from channel Y
Monitor access to port Z
Wait in reserve

A mission input command may take several hours to execute.

An input command to the mission level consists of a list of mission objectives or tasks. Each objective in the list has a "task frame" i.e. a database associated with the task with slots which define priorities, expected risks, costs, and time to complete. Each objective also has a list of required tools, and a list of preconditions that must be met before that objective can be attempted.

For example, a mission level input command may have a format of the following type:

```
List head - Mission (Name)
    Objective #1 (Name)
            Intended results (these include events, states, situations, etc.)
            Priority
            Acceptable risk
            Payoff for achieving
            Expected fuel cost
            Expected time to accomplish
            Required tools (these include weapons, ammunition, sensors, etc.)
                Reqtool #1 (Name)
                Reqtool #2 (Name)
            Preconditions ( these include events, states, situations, scenarios, etc.)
                Precondition #1
                Precondition #2
    Objective #2 (Name)
            Priority
            Acceptable risk
            Expected fuel cost
            etc.
```

Mission Level World Model

The mission level world model contains a list of conditions known to be true about the world and resources available to the vehicles and groups during the mission.

```
Resources List - Groups available
    Group #1 (Name, Type)
            List of vehicles assigned to group
            Group capabilities
```

- Speed
- Range
- Fuel
- Weapons
- Tools

Group #2 (Name, Type)
- List of vehicles assigned to group
  - Capabilities

Conditions List -- World State

Weather state
- Water current, direction and velocity
- Water temperature
- Water salinity
- Wave height
- Wave noise spectrum
- Turbidity (inverse of visibility)

Battle state
- At ease
- Blue alert
- Red alert
- Hostilities in progress
- Under fire
  - Attack
  - Retreat
  - Hide

Current map coordinates (or sectors) for
- Self
- Other team members

Current list of objects of attention
- Friendly forces
  - Aircraft
  - Surface Ships
  - Submarines
  - Mines
- Hostile forces
  - Aircraft
  - Surface Ships
  - Submarines
- Objects of interest to mission
  - Search sectors
  - Paths and waypoints
  - Landmarks

For each object on list
- Importance (mission value)
- Attributes
  - Attraction/Aversion (analogous to Trust/Fear)
  - Protect/Destroy (analogous to Love/Hate)

Task state
- Current task command being input at
  - Mission Level
  - Group Level (for each Group)

Vehicle Level (for each Vehicle)
E-move Level (for each subsystem)
Time on task and estimated time to completion for each
Resources expended and estimate to completion for each

Maps

The global database contains a world map in quadtree form as discussed in Section 7.1.1c. From that quadtree map, a local array map is derived for each hierarchical level, of scale such that the task at that level fits within the map boundaries. The mission level local array world map is illustrated in Figure 36. It covers an area of 64x64 kilometers, with 256x256 meter pixels.

On the mission map are geographical objects such as harbors, underwater mountains, canyons, coast lines, points of land, rivers, islands, navigation routes, navigation way points, and bottom depth contours. The mission map also contains estimated positions of mine fields, and positions and velocities of other battle groups, both friendly and unfriendly.

Each pixel on the quadtree map and mission level array map contain information such as:

Bottom depth
    Max, min, mean, standard deviation,
Map feature type (i.e. rock, hole, ship wreck, debris, etc.)
Object (vehicle, group, mine, ship, landmark, hazard, etc.)
Terrain type --
    Maximum slope
    Bottom character (rocks, sand, mud, etc.)

Overlaid on the mission map is a route graph with planned destinations, route markers, alternate destinations, and alternate route markers. Routes are stored as a graph of nodes, edges, and enclosed regions.

* Edges correspond to route segments. For each edge in the graph, the best available information is stored about route segment distance, traversibility, risk of detection, risk of destruction, landmarks visable along the route, etc.

* Nodes correspond to destinations, waypoints, intersections of routes, route markers, alternate destinations, and alternate route markers. Each node has a set of map coordinates

* Enclosed regions correspond to areas in which no route has been defined.

Since the nodes and way points of the route graph have map coordinates, the route graph can be overlaid on the mission map.

Mission Level Task Decomposition

At the mission level, mission objectives are decomposed into group tasks. The mission level task decomposition module therefore consists of a mission planner manager, and a planner and executor for each group.

The planner manager at the mission level assigns resources (vehicles, fuel, and weapons) and mission objectives to the group planners. It is the planner manager's responsibility to assure that the resources assigned to each group are adequate to accomplish the mission objectives with some safety margin.

The planner for each group is responsible for selecting a route for its group to follow. At the mission level, navigation is typically done by searching the route graph, not by searching the free space regions of the world map. No test needs to be made as to whether route graph edges intersect obstacles. The existence of an edge in the route graph can be assumed to guarantee that there exists a traversible path. The route graph gives the length of each route segment, the risk, and any other features that are relevant to the traversal of that route. The route graph is all that is necessary for mission level navigation planning. If enemy movements threaten particular routes, the route graph will be updated to reflect the change in risk associated with the affected route segments.

The planners for each group are also responsible for scheduling the list of objectives so as to maximize the expected mission score. The expected mission score is obtained by multiplying the probability of accomplishing each objective times the priority of that objective.

expected mission score = sum over i { pb(i) • pr(i) }

where pb(i) = probability of accomplishing objective(i)
pr(i) = priority of objective(i)

The actual mission score at any time t during the mission is obtained by multiplying the priority of each objective by the degree to which it was accomplished, and summing over the number of objectives accomplished up to time t.

actual mission score = sum over i { pr(i) • oa(i) }

where oa(i) = degree of accomplishing objective(i)

Both anticipated and actual mission scores are time dependent variables, which change as the mission evolves. The expected mission score ems(t) is the actual mission score up to time t, plus the expected mission score from time t until the end of the mission T.

ems(t) = ams(t) + ems(T-t)

where ems(t) is the expected mission score at time t

ams(t) is the actual mission score at time t

ems(T-t) is the expected mission score over the time interval between t and the end of the mission T.

ems(t) is similar to the evaluation function used in the A* search algorithm [47]. At t=0, the mission begins with prefabricated plans in place. As the mission evolves, the mission level group planners periodically replan the mission. At each replanning increment, a new plan will be generated by selecting the current plan that gives the best ems(t).

ems(t) will depend on enemy positions and intentions, and many other factors that may not be known before the mission and typically will change during the mission. The mission level planners must search a game tree in which the probable actions of the opponents and of nature must be estimated. Each node of the game tree is evaluated by computing an ems(t). The algorithms described in Section 7 are implemented to generate a plan graph for the mission executors.

As shown in Figure 27, a new mission level plan graph should be generated at least every 30 minutes. The mission level planning horizon is always the end of the mission.

Mission Level Executors

Each mission level executor executes the plan graph generated by its respective mission level planner. Output from the mission level executors are group task commands. Procedures called by the plan graph executors compute task command parameters such as required time to complete, degree of aggressive vs. conservative tactics to be used, degree of risk to be accepted, and payoff weights (priorities) for subtask achievements.

Mission executors branch to error recovery routines if difficulties arise. They call for emergency replanning action if a situation arises which is not covered in the existing plan.

Mission Level Sensory Processing

The function of sensory processing at the mission level is to integrate all data gathered during the entire mission with the a priori information provided before the mission began. It matches world model predictions with sensory observations. It detects conditions in the world that are different from what is in the model, and updates the world model. The identity and position of objects of interest in the world model are checked and verified by sensory data. Mission level sensory processing compares detected ocean floor features, GPS satellite navigation measurements, and inertial navigation system estimates with landmarks contained in the world model map. The results of these comparisons are integrated both spatially and temporally. This data is then used to update the world model, and to assign confidence levels to world model variables. New objects may be added to the interest list. Objects that have disappeared will be noted, and those that have been verified as destroyed will be so marked.

11.2 Level 5 - Group Control Level

The function of the group level is to decompose group task objectives into vehicle task commands. In the simplest case, a group consists of one vehicle. In all cases, group level input commands are decomposed into sequences of vehicle task commands for individual vehicles.

Inputs

Inputs to the group level consist of commands to perform group tasks in support of a scheduled mission objective. Examples of task commands to the group level are:

patrol
defend/attack sector
attack/evade group X
obtain intelligence on target list L

Commands may take several minutes to hours to carry out.

Group tasks must be selected from a task vocabulary, which is a set of group tasks for which there exists either a preprepared task plan, or an task schema from which a plan can be readily generated, or a search procedure for generating the task plan.

For each group task in the vocabulary, there exists a frame with slots which define required resources, preconditions, and constraints. There also exists a set of rules for partitioning the group task among the various vehicles within the group.

A typical group level input command has a format of the following type:

Current group task command (Name)
- Object of task
- Preconditions for task to begin
- Goal of task
- Resources needed to perform task
- Constraints (coordination, timing)
- Priority of task
- Acceptable risk
- Expected fuel cost
- Expected time to accomplish

List of possible next group tasks
- Anticipated task #1
    - Expected time till begin
    - Probability of being next task
    - Object of task
    - Preconditions for task to begin
    - Goal of task
    - Resources needed to perform task
    - Constraints (coordination, timing)
    - Priority of task
    - Acceptable risk
    - Expected fuel cost
    - Expected time to accomplish
    - Anticipated task #2

      .
      .
      .
      etc.

Group Level World Model

The group level world model contains a list of resources assigned to, or available to, the group.

Vehicles assigned to group (number)
- Vehicle #1 (Model, Serial #)
    - Capabilities
    - Speed
    - Range
    - Fuel
    - Weapons
    - Tools
- Vehicle #2 (Model, Serial #)
    - Capabilities

      .
      .
      .

The world model also contains :

Current map coordinates (or map sectors) for:
- Self group

Group center of mass
Group volume
Group perimeter
Group member positions

Current list of objects of attention
- Other groups
  - Friendly forces
    - Aircraft
    - Surface Ships
    - Submarines
    - Mines
  - Hostile forces

.
.
.

Priority for each
Attributes of each
Map coordinates of each

The group level world model also contains maps. For each group two data structures are extracted from the global database quadtree representation: 1) a 256x256 pixel group level world map centered about the group center-of-mass and scaled such that the commanded group task fits within the map boundaries, and 2) data for a 256x256 egospheric projection of the world map centered on the group center of mass. This data is provided to a real-time egosphere processor in the sensory processing system.

The group level world model contains a 256x256 array map of scale such that the current, and planned next, group task fits within the map boundaries. The group map shown in Figure 36 is a region of 8x8 kilometers, with a pixel resolution of 32x32 meters.

Each pixel in the map contains the following information:
- Bottom depth within the pixel -- (Max, min, mean, standard deviation)
- Map features contained in pixel -- (Type i.e. beach, gully, ridge, waypoint, landmark, etc.)
- Objects contained in pixel -- Other vehicles, known or suspected enemy positions, etc.
- Terrain type covered by pixel --
- Maximum slope
- Bottom type (sand, mud, rocks, etc.)

The world model also contains group level routes, stored as a graph of nodes, edges, and enclosed regions. The group level route graph defines planned destinations, routes, and alternate routes. For each route segment in the graph, there is an attribute list which contains the best available information about traversibility, risk of detection, risk of destruction, distance, of that route segment. If some information about a route segment is unknown, the route graph will indicate as much. If new information is learned during the mission, either via sensory measurements, or via communication from another source, it will be entered into the appropriate route segment attribute list.

For each route node in the graph there is an attribute list which indicates the node map coordinates, the node type, etc. Since the nodes and way points of the vehicle level route graphs have map coordinates, the group route graphs can be overlaid on the 256x256 pixel group world map.

The vehicle world model also contains a list of other groups and their attributes, such as position, velocity, composition, type, and capabilities such as speed, weapons, and range. Traces of group position are stored for a period extending about 50 minutes into the past.

The world model also contains information about states and the occurrence of events, such as subtask completion, or the appearance or disappearance of group level objects.

World model information contained in maps, group attributes, and historical traces is used by the group level planners to plan actions on, or manuevers relative to, objects that are part of a group. The current positions and velocities of objects within their group, and the timing of subtask completion events, are used by the executors for task sequencing.

World model information about groups of objects makes it possible to generate predictions about the position and motion of objects within that group. The world model generates such predictions for use by the group level sensory processing modules in the acquisition and interpretation of sensory data.

Group Level Task Decomposition

The function of the group level task decomposition module is to decompose group tasks into coordinated vehicle actions. The group level task decomposition module is analogous to a football quarterback, who analyses the defense and calls plays. The group task decomposition module attempts to recognize the tactics, capabilities, and intentions of opposing groups, to understand the risks and probable results of various actions, and to organize the activity of the group so as to maximize the score for each commanded group task.

The job assignment module of the group level planner manager contains an expert system which partitions the group task command into vehicle assignments. These are given to the group level task and route planners which generate plans for individual vehicles. The job assignment modules also assign resources such as fuel, sensors, and weapons to the vehicle planners. It is the job assignment module's responsibility to assure that the assigned resources are adequate for each vehicle to carry out its assigned activity with some safety margin.

Group Level Planners

The group level has a planner for each vehicle in the group. The route planning part of the group level planners correspond to the vehicle navigators. These navigators use processed data from instruments such as compasses, inertial reference systems, clocks, radar, and sonar, and correlate it with maps and charts. The job assignment manager commands the vehicle navigators to plan courses and routes based on group goals and objectives, fuel and time resources, obstacles, and potential threats.

The vehicle task schedulers and route planners may select predetermined, well practiced, and optimized coordinated cooperative vehicle maneuvers by naming the file in which they are stored. Coordinated cooperative vehicle maneuvers correspond to group tactics. Textbook group tactics may be obtained by referring to doctrine in tactics manuals.

Group tactics can also be computed, or recomputed, in real-time by gaming search strategies. The group level planners may make extensive use of real-time game theory techniques. As shown in Figure 27, the group level cyclic replanners generate new vehicle plans on the order of every 5 minutes. The planning horizon averages about 50 minutes into the future.

In order to facilitate planning, vehicle task commands carry lists of preconditions, resource requirements, expected costs, expenditure of resources, and risk factors.

The group level planner for each vehicle is responsible for selecting a route for each vehicle to follow, and for scheduling the list of vehicle tasks so as to maximize the expected group activity score. Evaluation functions are based on priorities, values of targets, risk factors, acceptable risk criteria, and probabilities of various outcomes. The expected group task score is obtained by multiplying the probability of accomplishing each vehicle task, times the priority of that task. The actual group score is obtained after the group task is ended by multiplying the priority of each vehicle subtask by the degree to which it was accomplished, and summing over the number of tasks.

expected group score = sum over i,j { pbv(i,j) • prv(i,j) }

where pbv(i j) = probability of vehicle(j) accomplishing subtask(i)

prv(i,j) = priority of vehicle(j) subtask(i)

actual group score = sum over i,j { prv(i,j) • oav(i,j) }

where oav(i,j) = degree of vehicle(j) accomplishing subtask(i)

Both expected and actual group scores are time dependent variables, which change as the group task evolves. Once the group task has begun, the expected group score is given by

egs(t) = ags(t) + egs(T-t)

where egs(t) is the expected group score at time t

ags(t) is the actual group score at time t

egs(T-t) is the expected group score over the time interval between t and the end of the group task T.

As the group task evolves, the group level vehicle planners will periodically replan the group task. At each t they select the plan that gives the best egs(t).

Group Level Executors

The group level executors execute the plans generated by the group level vehicle planners. Output from the group level executors are vehicle task commands. The group level executors are state sensitive expert systems that work from the set of IF/THEN state transition rules defined by the vehicle plan graphs. When feedback FB(5,s) indicates that a subgoal in the PST(5,s,tt) plan has been achieved, the executor EX(5,s) selects the next vehicle task command PST(5,s,tt+1) in the vehicle task plan. It then issues this planned command as an actual vehicle task command STX(5,s,t).

In all plans, whether prerecorded or computed in real-time, information about the state of the world can be used by the EX(5,s) executors to modify planned vehicle maneuver commands, to control decision points, to vary parameters such as speed, to effect synchronization and timing for cooperative coordinated movements and synchronized maneuvers between multiple vehicles.

Sensory Processing

Sensory processing at the group level compares measured navigational features such as buoys, bottom features, channels, points of land, rivers, and islands with information derived from the world model maps and lists of objects. Sensory processing also may detect and recognize synchronization signals and other messages from other groups cooperating on a mission plan.

At the group, an egosphere is used to compare sensory data with world map data. On the egosphere, the positions of objects are projected as they should appear to an observer in the group command vehicle, or group center-of-mass. This permits comparisons of sensory data with world map data, and facilitates the computation of directional maneuvers (turn toward or away from objects etc.). The types of objects projected onto the group egosphere are major terrain features such as ridges, gullies, and reefs, as well as other vehicles and known or suspected enemy positions.

As indicated in Figure 27, Kalman filters in the sensory processing system perform temporal integration over an interval extending about 50 minutes into the past. Attention is focused on objects and events that lie beyond the spatial and temporal range of interest for vehicle level sensory processing.

11.3 Level 4 - Vehicle Task Level

The vehicle task level receives commands from the group leader (at Level 5). Inputs command an individual vehicle to maneuver relative to some place, or target, or group of targets, possibly in cooperation with other members of the command group, and/or to execute a particular task, or sequence of tasks, in an environment containing multiple threats, obstacles, and unexpected hazards. Example input commands at the vehicle task level are:

enter harbor
dock with another vehicle
attack ship/sub
deploy mine
cut cable
inspect pipe
close/open valve

The vehicle task level also receives a set of priorities, values, and evaluation function parameters that allow world model predictions to be evaluated on the basis of estimates of cost, benefit, risk, and probability of success or failure. Vehicle task commands are expressed in terms of prioritized lists of targets and objectives. Target kill values and acceptable risk variables for each vehicle are specified by the vehicle task input commands.

It is the function of the vehicle task level to carry out the commanded vehicle tasks in such a way that the given set of priority values are maximized from a probabilistic standpoint.

Inputs

Inputs to the vehicle level consist of a command to perform a vehicle task in support of a group activity. This command will have the following format:

Vehicle task (Name)
- Object of task
- Preconditions for task to begin
- Goal of task
- Resources needed to perform task
- Constraints (coordination, timing)
- Priority of task
- Acceptable risk
- Expected fuel cost
- Expected time to accomplish

List of possible next vehicle tasks
- Anticipated task #1
    - Expected time till begin
    - Probability of being next task
- Anticipated task #2
    - Expected time till begin
    - Probability of being next
- Anticipated task #3
    - etc.

Vehicle tasks are selected from a task vocabulary. For each vehicle task in the vocabulary, there exists a frame with slots which define a list of required tools, a list of preconditions that must be met before the tasks can begin, and a procedure for decomposing the task. This procedure may consist of a prefabricated task plan, or an task schema from which a plan can be readily generated, or a search procedure for generating the task plan. There must also exist a set of rules for partitioning the vehicle task among the various subsystems within the vehicle. There may also exist a set of constraints that apply to subsystem assignments and vehicle task plans.

Vehicle Level World Model

The world model at the vehicle task level contains information defining the position and orientation of objects in the vicinity of the vehicle such as other vehicles, bottom features, buoys, piers, ships, mines, submarine nets, etc. This information is used by vehicle level planners to schedule E-Moves and compute E- Move parameters so as to plan attack or evasive maneuvers, and efficiently carry out vehicle task assignments.

Information about objects is indexed both by name of the objects and their position in space relative to the referenced vehicle. Attributes of objects such as their shape, size, velocity, type, condition, capabilities, probability of correct identification, and probable intended course of action are also contained in the world model at the vehicle task level.

The identity, position, and orientation of objects are used by the vehicle level task decomposition modules to plan and execute cooperative maneuvers relative to friendly objects or vehicles, and to plan and execute attack or evasion maneuvers relative to hostile objects or vehicles.

The vehicle level world model also contains a list of resources assigned to, or available to, the vehicle

List Head - Subsystems
- Subsystem #1 (Pilot)
  - Capabilities
    - Speed
    - Range
    - Fuel supply
- Subsystem #2 (Sensor suite)
  - Capabilities
  - Availability
  - Subsystem #3 (Communications)
  - Subsystem #4 (Weapons)
    - etc.

The vehicle level world model contains maps. For each vehicle task, two data structures are extracted from the quad tree mission level representation: 1) a 256x256 pixel map centered about the vehicle and scaled such that the vehicle task fits within the map boundaries, and 2) a egospheric projection centered on the vehicle.

In the 256x256 map, each pixel contains the following information:
Map features contained in pixel --
- Terrain type --
  - Maximum slope
  - Maximum and minimum depth
  - Standard deviation of depth
  - Bottom cover (rocks, sand, mud, etc.)

Pixel resolution is four meters and the entire array covers one square kilometer.

The group level 256x256 map also contains planned destinations, routes, and alternate routes. Vehicle level routes are stored as a graph of nodes, edges, and enclosed regions overlaid on the 256x256 pixel map. For each route segment, the best available information is stored about traversibility, risk of detection, risk of destruction, distance, etc. At the vehicle level and below, information may not be known a priori about the attributes of many route segments. As the vehicle explores a reqion with its sensors it will enter new information into the route graph. In particular, the position of local landmarks which may not be on a priori maps will be entered into the attributes of route segments as the vehicle moves and observes objects with its sensors. This information may be used by the same vehicle for later returning along the same route, or may be communicated to other vehicles for their use, or may simply be stored for subsequent missions.

The vehicle world model also contains a list of objects and their attributes, such as position, orientation, velocity, geometry, type, and capabilities such speed, weapons, and range. Traces of object position and orientation are stored for a period extending about ten minutes into the past.

The world model also contains information about states and the occurrence of events, such as subtask completion, or the appearance or disappearance of objects.

World model information contained in maps, object attributes, and historical traces is used by the vehicle level planners to plan actions on, or manuevers relative to, objects. The current positions and velocities of objects, and the timing of subtask completion events, are used by the executors for task sequencing.

World model information about objects makes it possible to generate predictions about the position and motion of objects. The world model generates such predictions for use by the vehicle level sensory processing modules in the interpretation of sensory data.

Vehicle Level Planner Manager

The planner manager at the vehicle task level is the vehicle captain. The captain receives commands from his respective group level executor EX(5,s), and interprets those commands in the context of what is present in the world model. The captain includes the job assignment module, the expert system that decides what jobs each of the vehicle subsystems should do to accomplish the vehicle level input command. The job assignment module examines the current state of the vehicle and its identified target, and issues job commands to the vehicle subsystem planners to generate plans for the type of maneuver to be performed relative to the target in order to achieve the commanded goal.

The vehicle planner manager module also selects the objective function to be used by the world model for computing the evaluation function EF(3,s).

Vehicle Level Subsystem Planners

The current MAUV system contains three subsystems. Plans for the next phase of the MAUV project, include a fourth subsystem, the weapons controller. For each subsystem there is a planner PL(4,s) and executor EX(4,s) module.

1. Pilot = {PL(4,1), EX(4,1)}
2. Sonar Controller = {PL(4,2), EX(4,2)}
3. Communications Controller = {PL(4,3), EX(4,3)}
4. Weapons Controller = {PL(4,5), EX(4,5)}

The vehicle planners PL(4,s) may select predetermined, well practiced, and optimized plans (i.e. E-move sequences) by simply naming the file in which they are stored. Generic plans, or scripts, can be selected by group technology codes, and E-move sequences can be computed in real-time by artificial intelligence planning and search strategies, by operational research linear programming techniques, or by game theoretic methods of cost-risk analysis and utility theory.

In order to facilitate planning, E-moves may carry lists of preconditions, resource requirements, expected costs, expenditure of resources, and risk factors. These parameters may either be specified as constants or as functions of world model state variables to be evaluated in real-time.

Planned coordination of E-moves between cooperating vehicle subsystems needed for vehicle maneuvers, manipulator motions, and sensor coordination, pointing, and focusing is organized and synchronized at this level. Synchronization can be carried out by including a timing field in the plans generated by the vehicle level E-move planners PL(4,s). The timing field may carry an <execute immediate> flag, a <begin on condition> flag, a <begin at clock time x> flag, a <begin after delay y> flag, a <begin with delay y after condition x> flag, an <end before clock time x> flag, a <do-until condition x> flag, or a <do-while condition y> flag.

1. Pilot

The pilot subsystem planner is responsible for generating trajectories, consisting of sequences of E-moves. These plans define maneuvers relative to target objects. The pilot planner schedules E-move piloting commands so as to maximize the expected vehicle task score. The vehicle maneuvers are scored on the basis of minimum risk and cost.

The pilot planner PL(4,1) selects E-moves from a vocabulary, or repertory of motion commands that the vehicle is capable of performing. The pilot planner never gives a command that is outside the vehicle's ability to perform. For each E-move in the vocabulary there exists a set of preconditions, required resources, constraints, and procedures for: flying the vehicle through the water, engaging an enemy in combat, cooperating with friendly vehicles, etc.

The pilot planner PL(4,1) is then responsible for planning an E- Move sequence according to evaluation functions provided by the vehicle level world model, and priorities defined in the vehicle task command. The evaluation functions EF(4,s) may cause the PL(4,s) planners to generate E-move sequences that satisfy least energy, or shortest time, or least risk, or maximum probability of success, or some other objective function. EF(4,s) may be a function of a large number of state variables.

2. Sonar

The sensor suite planner is responsible for selecting the proper settings for the sensors, and of planning the sensor pointing and scanning trajectories, and dwell time schedules to accomplish the commanded vehicle tasks.

Job commands JC(4,2) to the sonar controller planner PL(4,2) and executor EX(4,2) identify the target, the obstacle, the type of attack or surveillance tactic, and the evaluation criteria for generating the most effective and least risky (in terms of being discovered or targeted) sequence of sonar transmissions. This evaluation criteria may give kill value to the target, set limits on acceptable risk to the covertness or survival of the vehicle, etc.

3. Communications

The communications planner is responsible for formulating messages for communication, for computing the value of the information to be transmitted and its timeliness, and for deciding whether the value of the information exceeds the cost and risk of communicating.

Job commands JC(4,3) to the communications planner PL(4,3) and communications executor EX(4,3) identify the intended recipient of the message, the type of message, the evaluation criteria for generating the most effective and least risky sequence of communication transmissions. This evaluation criteria may give pay-off value to the information getting through, set limits on acceptable risk to the covertness or survival of the vehicle, etc.

4. Weapons

Job commands JC(4,4) to the weapons planner PL(4,4) and executor EX(4,4) identify the target, the type of attack, and the evaluation criteria for generating the most effective firing sequence. This evaluation criteria may place limits on ammunition expenditure, give kill value to the target, and set limits on acceptable risk to the self vehicle during the firing sequence.

The weapons planner PL(4,4) and weapons executor EX(4,4) are responsible for sequencing the arming, aiming, and firing of various weapons at specific targets. The weapons planner is responsible for determining the type of weapon, (torpedo, depth charge, etc.), fusing, the timing of firing, and the number of weapons to be expended against a particular target to achieve the commanded probability of kill.

The weapons planner receives commands from the same job assignment module JA(4,r) as the pilot. The pilot and weapons planner modules will therefore be expected to maximize their joint effectiveness by sharing information, and cooperating with each other in every way possible.

As the vehicle task evolves, the vehicle level E-move planners will periodically replan the vehicle task activity. As shown in Figure 27, the average replanning interval at the vehicle level is about one minute, and the planning horizon is about ten minutes.

Vehicle Level Executors

The vehicle level executors execute the plans generated by the vehicle level planners. Output from the vehicle level executors are subsystem commands to the E-move level.

In all plans, whether prerecorded or computed in real-time, information about the state of the world is be used by the EX(4,s) executors to modify planned E-move sequences, to control branches, to vary parameters such as speed, to effect synchronization and timing for cooperative coordinated movements and synchronized maneuvers between various subsystems.

Vehicle Level World Model

The world model at the vehicle level contains a 256x256 array local world map, on which are represented the position of the vehicle, the topology and characteristics of the bottom, and known objects. Pixel resolution is four meters and the entire array covers one square kilometer.

The vehicle world model also contains a list of objects and their attributes, such as position, orientation, velocity, geometry, type, and capabilities such speed, weapons, and range. Traces of object position and orientation are stored for a period extending about ten minutes into the past.

The world model also contains information about states and the occurrence of events, such as subtask completion, or the appearance or disappearance of objects.

World model information contained in maps, object attributes, and historical traces is used by the vehicle level planners to plan actions on, or manuevers relative to, objects. The current positions and velocities of objects, and the timing of subtask completion events, are used by the executors for task sequencing.

World model information about objects makes it possible to generate predictions about the position and motion of objects. The world model generates such predictions for use by the vehicle level sensory processing modules in the interpretation of sensory data.

Sensory Processing

Observed object features from the E-move sensory processing level enter the vehicle sensory processing modules where they are compared with predicted features from objects in the world model. Sensory processing at the vehicle level may use egospheres or world map coordinate frames to compare and integrate sensory data from a variety of sensors, and to match task level world model information against sensory observations. On the egosphere, the positions of objects can be projected such that directional maneuvers of the vehicle subsystems can be easily computed. The types of objects projected on the egosphere are major terrain features such as ridges, gullies, rivers, and navigation way points, as well as other vehicles, and known or suspected enemy positions.

Correlations and differences between predicted and observed sensory data are computed and integrated over a period of about one minute. If the integrated correlation is high, an object can be said to be recognized and a confidence factor in the attribute list of the predicted object in the world model is increased. If the integrated difference is high, the confidence factor in the world model is decreased. Once the world model confidence factor drops below threshold, the world model will be modified by updating with the new sensory data.

Vehicle level sensory processing modules recognize objects and temporal events such as the completion of E-move level commands. Sensory processing also may detect synchronization and identification signals and other messages from cooperating vehicles. Recognized object features and temporal events are entered into the vehicle task level world model, and passed upward to the sensory processing modules at the group level. Trajectories of objects and a history of temporal events are maintained over a period of about 10 minutes.

11.4 Level 3 - Elemental Move (E-move)

Level 3 decomposes elemental move commands (E-moves) into strings of intermediate poses which define motion pathways that are free of collisions.

Input commands consist of symbolic names of E-Moves. E-moves are typically defined in terms of motion of the subsystem being controlled through a space defined by a convenient coordinate system. E-move commands may consist of symbolic names of elemental movements which may be expressed as keyframe descriptions of desired relationships to be achieved between system state variables.

The term "keyframe" is derived from the field of cartoon animation. A keyframe in an animation sequence represents a particular relationship between the cartoon characters and objects in their environment at a key point in the story sequence. The keyframes define the story line, and are drawn by the principal artist and creator of the cartoon story. Intermediate frames are added by apprentice artists to fill in the action that connects the keyframes. A string of keyframes can thus be viewed as a string of goal poses to be achieved by the characters in the cartoon. The E-move level takes each successive keyframe goal as an input command, and generates the string of intermediate poses needed to smoothly move the system from each keyframe to the next.

Feedback inputs consist of best estimates of the position and orientation of features of objects such as bottom and obstacle surfaces, as well as edges, vertices, and holes of objects. Feedback is sampled every 600 milliseconds, but significant changes in feedback values occur on average only about once per every ten seconds.

Outputs consist of trajectories of intermediate poses that define motion pathways that move the controlled system from one keyframe pose to the next. The E-move level planners and executors perform the necessary computations to assure that the sequence of intermediate poses generated provides adequate clearance with potential obstacles.

Input commands

Input commands TC(3) to the E-move level call for an elemental movement or action designed to achieve some goal, typically a keyframe pose, defined in terms of a desired state in a specified coordinate frame. A keyframe pose for a pilot may consist of a desired vehicle position, orientation, and velocity to be achieved in a coordinate system of choice.

For the MAUV vehicles, pilot E-move commands are the following:

Go-to-pose(x, y, z, w, vx, vy, vz, vw, r, C)
where x, y, z define position, and
w defines orientation
of the vehicle at the keyframe goal pose.
vx, vy, vz define linear velocity, and
vw defines rotational velocity
at the keyframe goal pose.
r defines the distance from the goal pose at which the E-move is complete.
C defines the coordinate system of the E-move.

The coordinate system C may be defined in a target object, in which case, the goal keyframe pose defines the relative position between the vehicle and the target object.

Go-along-path(dx, dy, dz, dw, vx, vy, vz, vw, r, C)
where dx, dy, dz define the distance and
dw defines the rotation of the vehicle in the
keyframe goal pose relative to the current pose.
vx, vy, vz, vw define rates desired at the goal pose.
r defines the distance from the goal pose at which the
E-move is complete.
C defines the coordinate system of the current pose.

Turn-on-radius(ra, db, dx, dy, dz, v)
where ra is the radius of the turn
db is the desired change in bearing to be achieved by the turn.
dx defines the direction of the turn
dy defines the distance to the start of the turn
dz defines the vertical distance to be traveled per 90 degrees of turn
v defines the velocity to be maintained while turning

Spiral-on-radius(ra, db, dr, dx, dy, dz, v)
where ra is the starting radius

db is the desired change in bearing to be achieved during the spiral
dr is the rate of increase of radius per 90 degrees of turn
dx is the direction of the spiral
dy is the distance to the start of the spiral
dz is the vertical distance to be traveled per 90 degrees of turn
v defines the velocity to be maintained while turning

Follow-bottom-to(x, y, w, h, v, r, C)
where x, y are the map coordinates, and
w is the bearing at the goal pose.
h is the height above the bottom to be maintained.
v is the average velocity to be maintained
r is the distance from the goal pose at which the
E-move is complete.
C defines the coordinate system of the E-move.

Go-through(xr(i), yr(i), xl(i), yl(i), zt(i), zb(i),
x, y, z, w, v, r, C)
where xr(i), yr(i) define corridor markers to be kept on the right of the vehicle
xl(i), yl(i) define corridor markers to be kept on the left of the vehicle.
zt(i), zb(i) define top and bottom corridor bounds
x, y, z, w, define the goal pose
v defines the velocity to be maintained
r defines the distance from the goal pose at which the
E-move is complete.
C defines the coordinate system of the E-move

Continue(maneuver/coarse)

This command causes the vehicle pilot to continue the current maneuver, or the current coarse and bearing, until otherwise notified. It can be used to buy time when the Level 4 Vehicle/Task level requires an unexpected change in plans.

Stop
This command causes the vehicle to come to a stop as quickly as possible. It is an emergency command which can be issued when an unplanned halt in movement is required.

Sonar E-move commands consist of:

Scan-sector(xs,ys,xe,ye)
where xs, ys are the coordinates of the starting point
xe, ye are the coordinates of the ending point

Probe-direction(x, y, i, mode, ts)
where x, y is the direction to be probed
i is the sonar transducer identifier
mode = 1 denotes a single-read

mode = 2 denotes repeat-continuous
ts denotes the clock time sonar emission should begin

Communications E-move commands consist of:

Send-message(n, message, baud, power, ts)
where n denotes the number of characters in the message
message is a character string of length n
baud is the baud rate of the transmission
power is the transmitted power
ts denotes the clock time the transmission should begin

Task Decomposition - the H module

Planner Manager

For each subsystem at the E-move level, there is an E-move level planner manager module. Part of the planner manager is the job assignment modules JA(3,r), which selects the coordinate system most appropriate for computing the execution of the commanded E- move. The job assignment module also separates translational from rotational motions, and assigns the computation of intermediate positions and orientations to separate planners and executors for parallel computation.

E-move Planners

The E-move level planning modules PL(3,s) are responsible for generating a plan PST(3,s,tt) consisting of a problem-free sequence of intermediate poses that will accomplish the commanded E-Moves. The pilot planners check to see if there is adequate clearance between the vehicle and potential obstacles at all points along the planned path through in the world. If not, the planners interject additional intermediate poses so as to safely skirt potential problem areas.

The E-move pilot planning modules PL(3,s) add new intermediate trajectory points to the end of the current plan on average about as rapidly as the corresponding E-move executor selects points from the beginning of the plan to output to the Primitive level. Thus, the E-move planners generate an updated plan about every ten seconds, and the planning module typically has a plan prepared which looks at least two E-moves (or about 2 minutes) into the future.

E-move trajectories can be be planned in real-time as they are being executed, or can be preplanned and recorded. In a known environment, such in known waters or around known structures, commonly used E-move trajectories can be preplanned and stored as route graphs. Such preplanned trajectories can then be invoked by naming the file in which they are stored. Partially preplanned E-move trajectories can also be selected and then modified to fit the current environmental circumstances.

In unknown waters, or where known routes are not defined with enough resolution for E-move level planning, sensor-guided E-moves are required. In this situation, trajectory plans can only be made to the limit of the sensor range. Plans to avoid obstacles may employ hueristics designed to deviate as little as possible from higher level route plans.

E-move planning to engage enemy vehicles may employ a combination of set tactics and gaming techniques to devise the best sequence of manuevers given what is known about the state of the enemy, his intentions, and capabilities. The E-move level world model attempts to predict what the enemy state will be two minutes in the future (the E-move planning horizon). The planner then selects the action sequence for the next two minutes that will produce the best score vis-a-vie the predicted enemy state.

Of course, the enemy may not do what the world model predicts. In order to deal with this, the E-move level cyclic replanner generates a new plan every ten seconds. Every ten seconds the world model combines new sensor data with prior knowledge, and generates a modified prediction of enemy actions. The planner uses this modified prediction to generate a new plan which takes into account the new situation. If sensor data arrives which makes a current plan obsolete before the end of the ten second cyclic replanner interval, the emergency replanner begins immediately to generate a new plan.

Execution

The execution submodules EX(3,s) are responsible for issuing the first intermediate pose in the current plan to the appropriate task decomposition modules at the Primitive level. The execution submodule also monitors the progress of the Primitive level as it attempts to reach the commanded trajectory points. When a "Done" report is received from the Primitive level, the EX(3,s) module issues the next intermediate pose in the current plan.

Information from the world model about the current or anticipated future state of the world can be used by the executor EX(3,s) to modify planned E-move trajectories, to control branches, to vary parameters such as velocity, and to effect synchronization and timing for smooth trajectories. The executor can also respond immediately to emergencies. If sensory information indicates an emergency condition, the executor can switch immediately to preplanned emergency action until the emergency replanner can generate a new plan to deal with the situation.

Output from the E-move execution submodule consists of trajectory points, poses, and velocities in a coordinate system of choice. The output commands carry a field which designate the choice of coordinate system. Output strings of intermediate poses are not necessarily evenly distributed in time, but are chosen so as to steer the vehicle trajectories around problem areas such as obstacles.

World Model

The world model at the E-move level contains a 256x256 array local world map, on which are represented the position of the vehicle, the topology and characteristics of the bottom, and features (such as edges, vertices, and bounding surfaces) of known objects.

The E-move world model also contains a list of object features and their attributes, such as position, orientation, and velocity. Traces of feature position and orientation are stored for a period extending about two minutes into the past.

The world model also contains information about states and the occurrence of events, such as subtask completion, or the appearance or disappearance of object features.

World model information is used by the E-move level planners and executors to check clearances and perform local obstacle avoidance, and to define station-keeping poses, docking poses, and the aiming of sensors. The current positions and velocities of object features, and the timing of subtask completion events, are used by the executors for task sequencing. The historical traces are used by the planners.

World model information about object features makes it possible to generates predictions about the position and motion of sensed features, such as edges, corners, contours, etc. The world model generates such predictions for use by the E-move level sensory processing modules in the interpretation of sensory data.

Sensory Processing

Processed sensory data from the Primitive level enters the E-move sensory processing modules where it is compared with predicted data from the world model. The comparisons may be made in egosphere or world map coordinates. Correlations and differences between predicted and observed sensory data are computed and integrated over a period of about ten seconds. If the integrated correlation is high, a feature can be said to be recognized. If the integrated difference is high, the world model will be modified by updating with the sensory data.

E-move level sensory processing modules recognize object features such as edges, corners, and surfaces. Temporal events such as the completion of primitive level input commands are also recognized. Trajectories of object features and a history of temporal events are maintained over a period of about two minutes.

Recognized object features and temporal events are entered into the E-move world model, and passed upward to the sensory processing modules at the vehicle task level.

11.5 Level 2 - Primitive Level

The primitive level computes inertial dynamics, and generates smooth dynamically efficient trajectories in a convenient coordinate frame.

Command input consists of intermediate trajectory poses which define a path that has been checked for obstacles and is guaranteed free of collisions. Command input is updated on average once every ten seconds.

Feedback input consists of measured vehicle position, heading, velocity, rotation rates, rate of closure to obstacles and to the bottom. Feedback data at the primitive level is integrated over about a two second interval.

Output consists of evenly spaced trajectory points produced every two seconds. These trajectory points define a dynamically efficient movement. Delay between sensory data being sampled and output response from the Primitive level is two seconds.

Input Commands

Pilot primitive commands define desired vehicle poses at intermediate trajectory points in the coordinate system of choice. Vehicle Primitive commands are of the form:

Go-to(x, y, z, w, vx, vy, vz, vw, r, C)

where x, y, z define the desired position of the vehicle at end of the command.

w defines the desired yaw orientation of the vehicle at the end of the command.

vx, vy, vz, vw define the desired velocity at the end of the command. If vx, vy, vz are zero, the vehicle is required to stop at the commanded point. If the velocities are not zero, the vehicle should fly through the commanded point with the specified velocity.

r defines the tolerance to which the commanded point must be achieved. When the vehicle comes through a plane which contains the desired point and is perpendicular to the trajectory, the primitive level executor reports "done",

and the next primitive command is triggered.
C defines the coordinate system of choice

Continue(course)

This command causes the vehicle pilot to continue the current coarse and bearing, until otherwise notified. It can be used to buy time when the Level 3 E-move level requires an unexpected change in plans.

Stop

This command causes the vehicle to come to a stop as quickly as possible. It is an emergency command which can be issued when an unplanned halt in movement is required.

Task Decomposition - The H function

Planner Manager Modules

The job assignment modules at level 2 assigns the calculation of each coordinate variable to a separate process in order to facilitate parallel computation.

Plan Coordination Modules

The Plan Coordination Modules resolve constraints so as to guarantee that the coordinate variables produce the desired trajectories. Constraints on velocity, acceleration, braking, and jerk make dynamic trajectory coordination a complex problem.

Planner Modules

The primitive level planners PL(2,s) compute dynamically efficient trajectories between intermediate trajectory points defined by Primitive commands. Input commands define intermediate trajectory poses on the order of every ten seconds. The primitive level planners compute output command poses that are evenly spaced in time every two seconds.

Subcommands in the planned trajectories PST(2,s,tt) are synchronized so smooth coordinated motions of the vehicle are produced. The smoothness of planned trajectories can be controlled by limiting the jerk (third derivative of position) to a maximum value.

If the planned trajectories call for motions that transform into velocities or forces that exceed the physical limits of vehicle thrusters, the PL(2,s) planners must detect that condition, and scale back or modify planned trajectories PST(2,s,tt) so that the output subcommands to the level 1 servos are always within the range of capabilities of the servo level. This implies that the primitive level planners have access to a dynamic model which has the ability to compute the demands placed on the thrusters by commanded velocities and accelerations.

Executors

The planned trajectories PST(2,s,tt) from the planners PL(2,s) provide inputs to the executors EX(2,s). The primitive level executors EX(2,s) compare the current observed positions, velocities, and forces in the coordinate system of choice with the commanded (or desired) positions, velocities, and forces defined by the planned trajectories PST(2,s,tt). The errors between the desired plan PST(2,s,tt) and observed values FB(2,s,t) are used to compute outputs designed to achieve the desired values.

Output subcommands

Output subcommands from level 2, constitute input commands to level 1. Level 2 outputs define desired subsystem trajectories in the coordinate system of choice. Output subcommand values are updated every two seconds.

World Model

The world model at level 2 contains:

Current vehicle position, heading, velocity, and acceleration
A trace over the past ten seconds of position and heading
Current vehicle rotation rate, rotary acceleration
A trace over the past ten seconds of rotation
Vehicle mass and moments of inertia
Obstacle and goal points in vehicle centered world coordinates
Rate of closure of obstacle and goal points
A trace over past ten seconds of obstacle and goal points
Rate of approach to bottom
A trace of bottom distance over ten seconds

Current values from the world model are used by the executor modules to control the motion of the vehicle, and the pointing of the sonar and communications transponders. The traces integrated over the past ten seconds are used by the planner modules to generate plans for the next ten seconds. They are used by the sensory processing modules to filter and smooth incoming sensory data, and by the world model to predict future sensory data.

The primitive level world model also contains a dynamic model of the vehicle and its subsystems to be used in computing dynamically efficient trajectories.

Sensory Processing

Primitive level sensory processing modules operate on filtered data from level 1 from depth gages, bottom sensors, obstacle avoidance sonars, compass readings, and accelerometers. At the primitive level, sensory information is transformed into egosphere or world map coordinates. This permits data from a variety of sensors to be overlaid on a single map so that data from different sensors can be fused into a single representation of the position of the vehicle relative to measured points in the world.

The primitive level integrates information over a period of about two seconds. Information from a series of ten such integration periods is stored as traces, or trajectories. These can be used to calculate the motion of the vehicle relative to objects in the environment, or to compute rates of closure and intercept points.

Observed readings are compared with predictions from the world model. The sensory processing modules compute correlations and differences which are used by the world model to update the common memory. This updated information is used to compute better predictions for sensory processing, and to provide feedback to the planners and executors in the task decomposition module.

11.6 Level 1 -- Servo Level

Level 1 transforms coordinates from a convenient coordinate frame into joint coordinates, and servos joint positions, velocities, forces, and power.

Command input consists of commanded positions, velocities, thrust, power, orientation, and rotation rates of the vehicle, or of sensor subsystems in a coordinate system of choice. Command inputs are updated at regular intervals of two seconds.

Feedback inputs consist of measured rotation rates and torques of thrusters, measured compass headings, measured depth and altitude above bottom, measured range and bearing to objects including navigation transponders. Feedback inputs are sampled at regular intervals of 600 milliseconds.

Outputs consist of electrical voltages or currents to motors and actuators. These outputs appear 600 milliseconds after the inputs have been sampled.

Input commands

Input commands to level 1 are designated TC(1,r) r = 1,2, . . ., M, where M is the number of subsystems being controlled.

For vehicle thrusters and control surfaces, level 1 input commands TC(1,1) defines desired vehicle positions and orientations, velocities, and forces in a coordinate system of choice.

For sonar transducers, level 1 input commands TC(1,2) define frequencies, duration, and timing of sonic emissions, as well as desired pointing and tracking vectors.

For communication transducers, level 1 input commands TC(1,3) define frequencies, modulation, and timing of communication transmissions.

For the weapons system, level 1 input commands defines pointing and tracking vectors, and arming and firing commands.

Task Decomposition - The H module

The H module consists of Planner Manager, Planner, and Executor modules.

Planner Manager Module

The planner manager modules at level 1 perform kinematic coordinate transformations from a convenient coordinate system in which the control problem is most easily expressed, into joint coordinates.

There is a level 1 job assignment module for each vehicle subsystem: pilot (thrusters), sonar, communications, etc.

The level 1 Job Assignment modules must be able to work equally well with all coordinate systems of choice, and to switch readily back and forth between coordinate systems. The choice of coordinate system for each subsystem is made by the respective subsystem planner module at the Vehicle/Task level. At least four different coordinate systems may be selected. A coordinate system:
1) fixed in the vehicle,
2) fixed in the sensor platform,
3) fixed at a convenient point in inertial space,
4) fixed in an object of interest such as a submerged pipe or sunken ship hull.

Any of these coordinate systems may be either moving or stationary. For example, if a coordinate system is chosen fixed in a sunken hull of a ship, that hull may be swaying due to underwater currents.

Planner Modules

The servo level planners PL(1,s) interpolate (straight line, circular, or spline) desired thruster commands PST(1,s,tt) between level 1 command updates. These desired thruster commands provide smoothly varying inputs to the executors EX(1,s), one command for each 600 millisecond interval the feedback FB(1,s,t) is sampled.

The input commands TC(1,s) to the level 1 occur sufficiently frequently that each thruster planner can interpolate independently, and each thruster can be independently servoed to its respective plan.

Executor Modules

The level 1 executors EX(1,s) are servo controllers which compare the current observed thruster velocities and forces with the commanded (or planned) velocities and forces. The errors between planned and observed values are used to compute outputs designed to null the difference between planned and observed values. Plan and feedback inputs are sampled by the executors every 600 milliseconds.

Output subcommands

Output from the level 1 executor modules EX(1,s) consist of electrical voltages or currents. These outputs drive power amplifiers for thruster motors, sonar transducers, communication transducers, camera pan, tilt, zoom, focus, iris controls, etc.

Every control cycle, each level 1 executor selects a desired output from the plan generated by its respective level 1 planner, samples feedback represented in state variables in the world model, compares the desired value with the feedback, and computes an output. That output is written to an output register, and the executor waits for the next control cycle to begin. During the wait interval, a communications process moves new data into all level 1 input registers.

The time required at level 1 for the EX(1,s) modules to compute an updated output is 600 milliseconds.

World Modeling

The level 1 world model contains sensor readings that have been filtered and scaled to engineering units.

The world model at level 1 contains the best estimate of the current value of:
Thruster rpm for each thruster
Thruster force generated for each thruster
Range from each navigation transponder
Bearing of each navigation transponder
Range from each obstacle avoidance sonar
Bearing of each obstacle avoidance sonar
Depth
Compass heading
Altitude above bottom
Water temperature
System parameters (voltages, etc.)

This information is used by the servo level executors to compute the current output.

The world model also maintains a temporal trace over the past four seconds of the above information.

This may be used by the planner to generate a plan for the next four second interval. It may also be used by the world model itself to generate filter windows and to do Kalman filtering of the senory data variables so as to produce an improved estimate of their current value.
Common memory may also contain a priori information such as mass and moments of inertia of the vehicle and properties of the thruster blades.

Input to the level 1 world model comes from three sources:

1. From the task decomposition module
   - Task state information
2. From the sensory processing module
   - Detected, filtered, and scaled readings of sensors giving parameters such as range, altitude, etc.
   - Correlations and differences between observed and predicted sensor readings.
3. From a priori information loaded during system initialization.

Requests from the level 1 planners and executors to the world model module consist of Read-Requests for the value of named variables. Delay between executor requests and return of the information should be no more than a few bus cycles. At level 1, total loop delay for the vehicle control system, from sensory read, to executor output should be 600 milliseconds.

Sensory Processing

Level 1 sensory processing consists of scaling, filtering, and integration of individual sensor readings. Tachometer and accelerometer readings are transformed into velocities and accelerations. Joint encoders are processed into radians or degrees. Obstacle avoidance sensors are sampled and readings are converted into range in feet or meters, and bearing in radians or degrees. Navigation transponder readings are transformed into range and bearing. Compass readings are transformed into degrees, and depth and altitude sensors into feet or meters.

Data is filtered by placing an acceptability window around individual readings. Sensor readings that fall outside the window is discarded as spurious. If time permits, Kalman filtering may be performed on sensory data variables.

At level 1, emphasis is on short time delay. Data from the entire suite of sensors must be sampled, processed, entered into the world model, and then accessed and used by the servo level executor modules in one executor clock cycle. Since the acoustic delays in each sonar transducer can easily be 20 milliseconds and there are several such sensors, timing is critical. Typically sensor readings are synchronized with the executor clock so as to minimize time delays between sampling and acting on the sampled data.

The input to the level 1 sensory processing module corresponds to measurements of points in the environment, or in state space. Output consists of filtered trajectories in space and time.

12. Summary and Conclusions

The MAUV project has made good progress toward its objective of demonstrating intelligent cooperative behavior in multiple autonomous undersea vehicles. Much has been learned about how to build a control system architecture which fully integrates concepts of artificial intelligence and game theory with those of modern control theory. A control system architecture has been defined that can enable a team of cooperating intelligent vehicles to compete against a team of cooperating intelligent opponents in a real-time dynamic environment.

12.1 Progress to Date

* A Real-time Control System (RCS-3) with an open system architecture has been designed with six hierarchical layers of task decomposition, world modeling, and sensory processing. Functionality is defined and code written at all six levels. Each module in the system has a clearly specified function, and well defined I/O interfaces. Data flow and timing are specified.

* A formal representation of real-time planning has been developed, using game theory and value driven logic

* A conceptual design for dynamic world modeling has been developed, using multi-dimensional world maps and a real-time object oriented database

* A new approach to sensory data fusion has been developed, using egosphere representations, real-time model matching, and stereo/motion integration.

* Two autonomous underwater vehicles of EAVE-EAST design have been constructed and equipped with a five beam obstacle avoidance sonar, altitude and depth sonar, an acoustic navigation system, pressure and temperature sensors, a communications system, a hierarchical control system, and intelligent software.

* A six level RCS-3 control system architecture has been installed on the EAVE-EAST vehicles. Code at the lowest three levels was integrated and tested on the vehicles in Lake Winnipesaukee, and code at the highest level was run in simulation.

* A real-time multi-processor computer system was designed and constructed consisting of five CPUs per vehicle. This system uses a commercial (pSOS) real-time operating system with multi-tasking and multi-processors. The hardware and operating system are capable of running both C and Lisp simultaneously with real- time communications between the C and Lisp programs.

* A network of 13 SUN computers was procured and assembled into a program development environment running under UNIX.

* Two sets of target computer hardware were constructed and integrated into two vehicles. The hardware consists of five 68020 computer boards, four megabytes of RAM, and 400 megabytes of mass storage using optical disk technology.

* A underwater environmental simulator for two autonomous vehicles was developed. Three versions were coded and installed to run a SUN, on a micro-VAX, and on the vehicle target hardware.

Funding during FY87 was $2.3 million. Total funding for the entire MAUV project was $4.0 million.

12.2 What Remains To Be Done

12.2.1 Control System Development

Despite great progress, much remains to be done on the control system development. The complete six level NBS hierarchical control system has not yet been fully implemented, tested, integrated, and demonstrated, in both simulation and on the MAUV vehicles. All of the hierarchy levels do not yet exhibit real-time planning and world model updates. The current MAUV programming and debugging environment still needs to be enhanced.

Acoustic communications have not yet been fully implemented between the two vehicles and the base station. Communications protocols that weigh the value of the communicated information against the risk of breaking communications silence need to be implemented.

There are still many issues of multivehicle command, control, and communication that still need to be addressed. Methods for transmitting commands with limited bandwidth, and with risks associated with communication emissions, need to be explored. The question of how to maintain cooperative group behavior when communications are lost needs much more study.

12.2.2 Cooperative Search and Map Demo

The cooperative search and map scenario has yet to be implemented in a gaming environment against both fixed and moving simulated defenses. The purpose of the search and map scenario is to illustrate the ability of two simulated MAUV vehicles to transit, fly formation, penetrate a defensive barrier, and execute a coordinated search pattern.

The defensive barrier will consist of both fixed and moving defenses. A gaming environment will permit the moving defenses to be controlled by humans at gaming screens that display the information available to them about the whereabouts of the simulated MAUV vehicles. The vehicles controlled by the human players will have simulated weapons which they can use in attempting to prevent the MAUV vehicles from accomplishing their missions.

This gaming environment will permit testing and evaluation of the same control system hardware and software that will be used to operate the real UNH EAVE-EAST MAUV vehicles in Lake Winnipesaukee to demonstrate real search capabilities.

12.2.3 Cooperative Search and Attack Demo

The cooperative search and attack scenario also has not yet been implemented. The purpose of this demo is to illustrate the ability of the two simulated MAUV vehicles in a gaming environment to transit, fly formation, execute a coordinated search pattern, and upon finding a target, to carry out a coordinated attack against it.

In the search and attack scenario, human players would control a simulated target vehicle and defensive vehicle. The human players would attempt to get the target vehicle safely through a channel where the the MAUVs have established a barrier. The MAUVs can be equipped with simulated weapons, and attempt to prevent the target from passing, by destroying it, or by forcing it into a fixed obstacle such as a mine field or shoal.

12.2.4 Advanced Simulator/Gaming Environment

A next generation simulator/gaming environment system needs to be designed and implemented in order to accomplish the games outlined above. Also, design requirements and specifications need to be developed for an advanced MAUV simulator/gaming environment. The advanced system should include the ability to accommodate up to ten MAUV vehicles, as well as multiple human operators controlling a variety of submarine vehicles, sensors, surface ships, aircraft, and missiles. A system similar to the SIMNET tank warfare simulation system should eventually be developed for multiple autonomous undersea vehicles.

12.2.5 Transfer of MAUV Control System to MK-30 Vehicle

Discussions have taken place with the Naval Underwater Systems Center at New Port, Rhode Island, to adapt a MAUV RCS-3 control system for installation of a MK-30 target vehicle. If this were done, NBS would provide the control system software. NBS would also work with NUSC to install the MAUV control architecture on the MK-30, and to test, and evaluate the performance of the MK-30 using the MAUV system.

12.2.6 Visual Bottom Following and Mapping

The MAUV vehicles still need to be equipped with high resolution sensors, including a side scan and forward scanning sonar system, and a vision system. The first vision system would consists of a TV camera and a structured light projector. This would permit a plane of light to be projected on the bottom, and the image of the reflected light stripe to be analyzed on a 68020 computer on-board the vehicle to generate information about the shape and distance of the bottom. This 3-D information could then be inserted into the world model where it would be used by the control system to generate bottom-following commands.

A light projector on one vehicle could also be used to illuminate a target while the second vehicle photographs it. Both flood lighting and light striping should be demonstrated. An on-board video tape recorder can be used to store images from the camera for processing through an on-shore Pipeline Image Processing Engine (PIPE).

12.2.7 Real-time 3-D Vision

A feasibility demonstration model of a passive 3-dimensional real-time underwater vision system also needs to be designed, constructed, and tested. Methodologies based on stereo and image flow can be demonstrated using PIPE. A new technique for extracting range from brightness also should be tested [48].

Egosphere representations need to be developed for building topological maps of the bottom, and for matching visual and sonar data with them.

12.3 Transfer of MAUV Technology

In December 1987, the DARPA Office of Naval Technology made a decision to terminate the MAUV project due to lack of funding. This decision was coupled with a directive by DARPA management to attempt to transfer MAUV technology to other DARPA projects. There appear to be many potential applications.

The MAUV RCS-3 open system architecture might serve as a standard reference model for the specification and design of intelligent control systems for a great many autonomous vehicle projects, not only in DARPA, but for the Air Force, Navy, Army, and Marine Corps as well. For example, the Army TEAM project for multiple semi-autonomous land vehicles has chosen RCS-3 as a control system architecture.

RCS-3 has also been applied to battle management for SDI and has potential applications for other battle management systems. Equally important, RCS-3 might be used to define interfaces between groups of autonomous (or semi-autonomous vehicles) and higher level battle management systems.

The RCS-3 control architecture is applicable to robot manipulators as well as to autonomous vehicles. It accommodates both autonomous and teleoperated systems. The NASREM architecture [6] developed for the NASA space station telerobot system is a version of RCS-3. The NASREM architecture has also been selected by FMC as the control architecture for the DARPA funded ARM project.

The RCS-3 open system control architecture is still under active development. There remain many issues which need further study and development. Nevertheless, RCS-3 has already been shown to have many applications, and appears to have great potential for many others.

13. Acknowledgments

The MAUV project was funded by the Defense Advanced Projects Agency, Office of Naval Technology, Dr. Ron Clark, Director. Mr. Charles Stuart was the DARPA project sponsor.

At the National Bureau of Standards, Maris Juberts headed the systems integration team, Marty Herman and Scott Swetz headed the real-time planning team, Tsai-Hong Hong headed the world modeling team, Tom Wheatley was responsible for communications and timing, John Michaloski headed the programming environments team, Peter Mansbach headed the operating systems team, A. Abrishamian headed the sensory processing team, Mike Ali headed the executor team, Marty Roche headed the simulation team, Don Orser headed the theoretical foundations team, and Ken Goodwin provided management support. Also working on the MAUV project in a variety of capacities were Rick Quintero, Tsung Ming Tsai, Hui-Min Huang, Steve Legowik, Su Chen Chang, Dave Oskard, Barry Warsaw, Mark Rosol, Roger Bostelman, Wendell Wallace, and Bob Russell. Jim Albus was the project manager.

At the University of New Hampshire, Richard Blidberg was the principal investigator, Jim Jalbert was the project leader, Steve Chappell headed software systems, Mike Shevenell headed hardware/software integration, Dennis Stamulis was responsible for the operation system, Bob Welsh headed electronic systems, and Rod Haywood headed mechanical systems.

Also involved in the MAUV project was Professor Allen Waxman, of Boston University, who performed research on stereo vision for AUVs using the NBS Pipeline Image Processing Engine (PIPE). University of Maryland under Professor Azriel Rosenfeld conducted experiments on depth from image flow in the underwater environment, also using PIPE. Lehigh University under Professors Roger Nagel and Glen Blank did studies of programming techniques for RCS-3 using state-graph techniques.

At Decision Science Applications, George Pugh provided expertise in value driven logic and Joe Krupp developed the mission level planner. Martin Marietta Baltimore provided an environmental simulator for scenario development. Bob Finkelstein, of Robotic Technology, Incorporated, did scenario development and performance evaluation techniques for MAUV demonstrations.

Products named in this report are listed for purposes of information only. There is no implied endorsement of any products or implication that they are the best available for the purpose.

REFERENCES

[1] D. R. Blidberg, "Guidance Control Architecture for the EAVE Vehicle", IEEE Journal of Oceanic Engineering, October 1986.

[2] J.S. Albus and D.R. Blidberg, "A Control System Architecture for Multiple Autonomous Vehicles (MAUV)", Proceedings of the Fifth International Symposium on Unmanned, Untethered Submersible Technology, University of New Hampshire, June 22-24, 1987.

[3] A.J. Barbera, J.S. Albus, M.L. Fitzgerald, and L.S., Haynes, "RCS: The NBS Real-time Control System", Robots 8 Conference and Exposition, Detroit, MI, June 1984.

[4] J.A. Simpson, R.J. Hocken, and J.S. Albus, "The Automated Manufacturing Research Facility of the National Bureau of Standards", Journal of Manufacturing Systems, Vol.1, No. 1, pg. 17, 1983.

[5] J.S. Albus, "A Control System Architecture for Multiple Autonomous Undersea Vehicles (MAUV)", The Fourteenth Annual AUVS Technical Symposium and Exhibit, Washington, DC July 19-21, 1987.

[6] J.S. Albus, H.G. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)", National Bureau of Standards Technical Note 1235, July 1985.

[7] J. Lowrie, et al. "Autonomous Land Vehicle", Annual Report, ETL-0413, Martin Marietta Denver Aerospace, July 1986.

[8] J. Graham, "Intelligent Task Automation (Phase II) Sixth Quarter Quarterly Report", Martin Marietta Denver Aerospace, March 1987.

[9] T.B. Sheridan, "Supervisory Control of Remote Manipulators, Vehicles and Dynamic Processes", In Rouse, W.B. (Ed.) Advances in Man-Machine Systems Research, Vol. 1, NY JAI Press, pgs 49-137, 1984.

[10] J.S. Albus, C. McLean, A.J. Barbera, and M.L. Fitzgerald, "An Architecture for Real-time Sensory-Interactive Control of Robots in a Manufacturing Environment", 4th IFAC/IFIP Symposium on Information Control Problems in Manufacturing Technology, Gaithersburg, MD, October 1982.

[11] C. McLean, M. Mitchell, and E. Barkmeyer, "A Computer Architecture for Small Batch Manufacturing", IEEE Spectrum, pg. 59, 1983.

[12] J.S. Albus, "Brains, Behavior, and Robotics", BYTE/McGraw-Hill, Petersborough, NH, 1981.

[13] M.O. Shneier, E.W. Kent, and P. Mansbach, "Representing Workspace and Model Knowledge for a Robot with Mobile Sensors", Proceedings of the 7th International Conference on Pattern Recognition, pg. 199, 1984.

[14] W.A. Perkins, "A Model Based Vision System for Sensor-Controlled Manipulation and Inspection", IEEE Transactions on Computers, Vol. C-27, pg. 126, 1978.

[15] G.L. Gleason, and G.J. Agin, "A Modular Vision System for Sensor-controlled Manipulation and Inspection", Proceedings of the 9th International Symposium on Industrial Robots, pg. 57, 1979.

[16] R.C. Bolles, P. Horaud, and M.J. Hannah, "3DPO: Three Dimensional Parts Orientation System," Proceedings of the International Joint Conference on Artificial Intelligence, pg. 1116, August 1983.

[17] T.F. Knoll and R.C. Jain, "Recognizing Partially Visible Objects Using Feature Indexed Hypotheses", Proceedings of the IEEE Conference on Robotics and Automation, San Francisco, CA., pg. 925, April 1986.

[18] B. Hayes-Roth, "A Blackboard Architecture for Control, Artificial Intelligence (1985), pgs. 152-321.

[19] A. Barr, and E. Feigenbaum, The Handbook of Artificial Intelligence, (Los Altos, William Kaufman, 1981).

[20] M. Brady, et.al, ed. Robot Motion: Planning and Control, (Cambridge, MIT press, 1982).

[21] D.E. Whitney, "Resolved Motion Rate Control of Manipulators and Human Prostheses", IEEE Transactions Man-Machine Systems MMS-10, pg. 47, 1969.

[22] R.P. Paul, Robot Manipulators: Mathematics, Programming and Control, (Cambridge, MIT Press, 1981).

[23] J.C. Crowley, "Navigation for an Intelligent Mobile Robot", IEEE Journal of Robotics and Automation, Vol RA-1, No. 1, pg. 31, 1985.

[24] G.E. Pugh and D.F. Noble, "An Information Fusion System for Wargaming and Information Warfare Applications", Decision-Science Applications, Inc., Report No. 314, May 1981.

[25] G.E. Pugh and G.L. Lucas, "Application of Value-Driven Decision Theory to the Control and Coordination of Advanced Tactical Air Control Systems", Decision-Science Applications, Inc., Report No. 218, April 1980.

[26] S.G. Chappell, "Blackboard Based System for Context Sensitive Mission Planning in an Autonomous Vehicle", Proceedings of the Fifth International Symposium on Unmanned, Untethered Submersible Technology, University of New Hampshire, June 22-24, 1987.

[27] A. Waxman, "Convected Activation Profiles and Image Flow Extraction", Laboratory for Sensory Robotics, College of Engineering, Boston University, LSR-TR-4, Boston, MA.

[28] A. Waxman and J. Duncan, "Binocular Image Flows: Steps toward Stereo- Motion Fusion", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8, No. 6, November 1986.

[29] G.D. Blank, "Responsive System Control Using Register Vector Grammar", CSEE Department, Lehigh University, Bethlehem, PA.

[30] G.E. Pugh and J. Krupp, "The Control of Autonomous Underwater Vehicles through a Hierarchical Structure of Value Priorities", Proceedings of the Fifth International Symposium on Unmanned, Untethered Submersible Technology, University of New Hampshire, June 22-24, 1987.

[31] Martin Marietta Contract 50SBNB6C4125 Progress Reports.

[32] R. Finkelstein, MAUV Final Report, "Measures of Performance and Effectiveness for the MAUV Project", October 1987.

[33] M. Daily, J. Harris, D. Keirsey, K. Olin, D. Payton, K. Reiser, J. Rosenblatt, D. Tseng, and V. Wong, "Autonomous Cross-Country Navigation with the ALV", Proceedings DARPA Knowledge-Based Planning Workshop, Austin, TX, pgs. 20-1 - 20-10, December 1987.

[34] D.W. Payton, "An Architecture for Reflexive Autonomous Vehicle Control", Proceedings IEEE International Conference on Robotics and Automation, San Francisco, CA pgs. 1838-1845, April 1986.

[35] J. Lowrie, R. Greunke, M. Thomas, B. Gothard, R. Koenig, G. Celvi, and R. Rehn, "Autonomous Land Vehicle", ETL-0413, Engineer Topographic Laboratories, Fort Belvoir, VA, July 1986.

[36] S.A. Shafer, A. Stentz, and C.E. Thorpe, "An Architecture for Sensor Fusion in a Mobile Robot", Proceedings IEEE International Conference on Robotics and Automation, San Francisco, CA, pgs. 2002-2011, April 1986.

[37] C. Isik and A. Meystel, "Decision Making at a Level of a Hierarchical Control for Unmanned Robot", Proceedings IEEE International Conference on Robotics and Automation, San Francisco, CA, pgs.1772-1778, April 1986.

[38} C. McLean, H. Bloom, and T. Hopp, "The Virtual Manufacturing Cell", IFAC/IFIP Conference on Information Control Problems in Manufacturing Technology, Gaithersburg, MD, October 1982.

[39] J.S. Albus, E.W. Kent, M. Nashman, P. Mansbach, L. Palombo, and M. Shneier, "Six-Dimensional Vision System", Proceedings of the 12th International Symposium on Industrial Robots, Paris, France, June 1982.

[40] J.S. Albus, "Mechanisms of Planning and Problem Solving in the Brain", Mathematical Biosciences, Vol. 45, pgs. 247-293, August 1979.

[41] W.T. Miller, F.H. Glanz, and L.G. Kraft, "Real Time Dynamic Control of an Industrial Manipulator Using a Neural Network Based Learning Controller", IEEE Journal of Robotics and Automation, February 1988.

[42] M. Herman and J.S. Albus, "Real-time Hierarchical Planning for Multiple Mobile Robots", Darpa Knowledge-Based Planning Workshop, Austin, TX December 8-10, 1987.

[43] D.N. Oskard and T.H. Hong, "A Spatial Mapping System for Autonomous Vehicles", NBS Technical Report (in process), April 1988.

[44] S. Chen, "Multisensor Fusion and Navigation of Mobile Robots", Department of Computer Science, University of Maryland Computer Science Department, University of North Carolina, Charlotte, NC.

[45] R.C. Nelson and J. Aloimonos, "Finding Motion Parameters for Spherical Flow Fields", University of Maryland Computer Science Department, TR-1840, April 1987.

[46] L. Conway, R. Volz, and M. Walker, "New Concepts in Tele-Autonomous Systems", Robotics Research Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI.

[47] N.J. Nilsson, Problem-Solving Methods in Artifical Intelligence, McGraw-Hill, New York, NY, 1971.

[48] AMELEX American Electronics, Inc. Final Report, "3-D Robotic Vision Using Structured Lights and Intensity Measurements", Purchase Order 43NANB716798, February 1988.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NIST/SP-1251	2. Performing Organ. Report No.	3. Publication Date September 1988

4. TITLE AND SUBTITLE

System Description and Design Architecture for Multiple Autonomous Undersea Vehicles Project

5. AUTHOR(S)

James S. Albus

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
(formerly NATIONAL BUREAU OF STANDARDS)
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

DARPA Defense Advanced Research Projects Agency
Office of Naval Technology
1400 Wilson Blvd.
Arlington, VA 22209-2308

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

The objective of the MAUV project is to demonstrate intelligent cooperative behavior in multiple autonomous undersea vehicles.

The approach is to build a control system architecture which will fully integrate concepts of artificial intelligence and game theory with those of modern control theory. The control system is being designed to permit a team of cooperating intelligent vehicles to compete against a team of cooperating intelligent opponents in a real-time dynamic environment.

Among the significant technologies being pursued are:

* Real-time planning, using game theory and value driven logic
* Dynamic world modeling, using multi-dimensional world maps and a real-time object oriented database
* Sensory data fusion, using egosphere representations, real-time model matching, and stereo/motion integration
* Multiplayer gaming

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

dynamic worldmodeling; egosphere representations; multiplayer gaming; multi-dimensional world maps; real-time planning; sensory data fusion; value driven logic

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

126

15. Price

USCOMM-DC 6043-P80

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
(FORMERLY NATIONAL BUREAU OF STANDARDS)
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

Official Business
Penalty for Private Use $300

Stimulating America's Progress
1913-1988