

Communications

Three-Dimensional Visualization of Mission Planning and Control For the NPS Autonomous Underwater Vehicle

MICHAEL J. ZYDA, ROBERT B. MCGHEE, FELLOW, IEEE,
SEHUNG KWAK, MEMBER, IEEE, DOUGLAS B. NORDMAN,
RAY C. ROGERS, AND DAVID MARCO

Abstract—Unmanned vehicles can operate where humans cannot or do not want to go. The last decade's advances in computer processor capability and speed, component miniaturization, signal processing, and high-energy density power supplies have made remotely operated vehicles (ROV's) and, to some extent, autonomous vehicles, a reality. In an effort to further advance this technology, the Naval Postgraduate School (NPS) is constructing a small autonomous underwater vehicle (AUV) with an onboard mission control computer. The mission controller software for this vehicle is a knowledge-based artificial intelligence (AI) system requiring thorough analysis and testing before the AUV is operational. We discuss how rapid prototyping of this software has been demonstrated by developing controller code on a LISP machine and using an Ethernet link with a graphics workstation to simulate the controller's environment. Additionally, we discuss the development of a new testing simulator using a KEE expert system shell that is designed to examine AUV controller subsystems and vehicle models before integrating them with the full AUV for its test environment missions. This AUV simulator utilizes an interactive Mission Planning Control Console and is fully autonomous once initial parameters are selected.

I. INTRODUCTION

Autonomous vehicles can go where humans cannot or do not want to go. Autonomous vehicles are capable of receiving initial input, moving to another location, executing a mission, and returning with the requested results or data. In addition to performing labor intensive or repetitive tasks, these vehicles can perform their jobs faster and with greater precision than humans, and can also proceed into hostile or contaminated environments.

For the last 30 years, remotely operated vehicles (ROV's) have attempted to fill these needs and the last decade's tremendous advances in computer and systems engineering, as well as the rapid deployment of ROV's, here validated vehicle and sensor designs. AUV design is not so advanced. While ROV's can cheaply use rapid prototyping and testing techniques, AUV's are still quite complicated and costly. The operational testing process subjects these expensive vehicles to harsh and unpredictable environments in which the logistics are difficult and some AUV losses are unavoidable. While there is no substitute for operational experience, most of the AUV's engineering problems have been solved while learning how to operate ROV's. The last step is the development of high-level mission planners that can be simply and cheaply tested without expensive logistics and unaffordable losses.

One solution to this problem is through the use of three-dimensional visual simulation. Through the use of vehicle simulators, AI mission planners can be developed in the laboratory and can

receive data inputs from artificial (computer-generated) sources. The planner's outputs can be used to drive a simulator whose actions can be observed and interpreted to evaluate the effectiveness of the planner without having to risk the vehicle. The same powerful computer systems that have revived AUV research can thus be used for rapid AUV prototyping and low-risk initial testing.

The U.S. Navy has identified a number of tasks that can be performed by AUV's and the Defense Advanced Research Projects Agency (DARPA) strongly supports AUV research [1], [2]. The Naval Postgraduate School (NPS) is developing an experimental AUV to support research relating to these military requirements. Part of this project is the design of three-dimensional visual simulators that will reduce the time and expense of implementing various AUV subsystems, while also permitting efforts to proceed along several simultaneous approaches. Previous simulator research at the NPS [3] has shown that graphics workstations provide a useful way to simulate a realistic external environment for conducting AUV operations, and more recently, a new simulator has been developed to generate a "laboratory environment" for testing several AUV planning, navigation, and control subsystems [4]. This approach permits the prompt development and thorough testing of AI software for the NPS AUV, the examination of different AUV hydrodynamic models, the testing of maneuvering subsystems in conjunction with different sensor configurations, as well as the prototyping and development of AUV Mission Planning Control Stations for possible use on AUV launching platforms. These proven subsystems are being integrated with the AUV mission planner to develop full-scale proposed missions before operational testing of the actual vehicle.

II. NPS AUV SIMULATOR DEVELOPMENT HISTORY

A. Vehicle Characteristics

The NPS AUV is modeled after the Swimmer Delivery Vehicle (SDV) used for the delivery and extraction of the U.S. Navy Special Warfare Teams. The actual NPS Model 2 AUV resembles the SDV and the simulator's vehicle dynamics have been scaled to the dimensions of the AUV currently under construction.

The simulator's controller carries out AUV operations by directing its output to either of two three-dimensional visual representation systems. NPS AUV-Sim1 is a simplified vehicle which models complex AUV missions in the open-ocean environment. NPS AUV-Sim2 uses a more sophisticated SDV hydrodynamic model in a small "test pool" to evaluate various AUV configurations and to develop the actual control algorithms that are being used by the NPS Model 2 AUV.

1) *NPS AUV-Sim1*: The first three-dimensional vehicle simulation system [3] permits mission execution without requiring a detailed implementation of AUV dynamics. The simulator represents a small manned vehicle with a control panel and a "through the periscope" display similar to that of the U.S. Navy's Sturgeon class attack submarine. The vehicle has a single screw and rudder and maintains continuous neutral buoyancy. Aft-mounted sternplanes impart a hull pitch angle for large depth changes, while forward-mounted bowplanes provide more precise depth control without generating a pitch angle. Although operators can manually operate the AUV, the vehicle is normally under autopilot control.

The NPS AUV-Sim1 dynamics model consists of a simple point-

Manuscript received October 8, 1989; revised March 27, 1990. This work was supported by the Naval Postgraduate School's Direct Funding Program.

The authors are with the Department of Computer Science, Code 52, Naval Postgraduate School, Monterey, CA 93943.

IEEE Log Number 9036197.

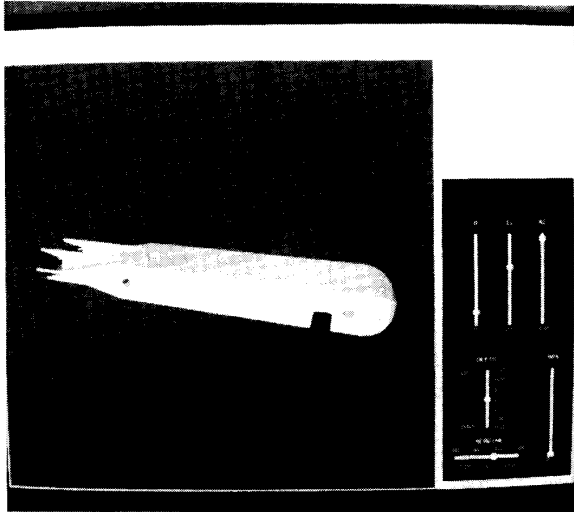


Fig. 1. NPS AUV-SIM2.

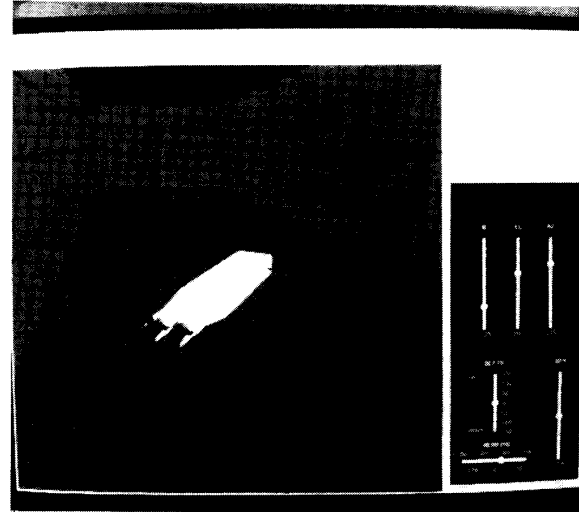


Fig. 2. NPS AUV-SIM2.

mass approximation governed by one acceleration equation, two rate equations, and one attitude equation. The vehicle's location and orientation is described by applying these equations at a 10-Hz rate and by setting the autopilot's control surface positions according to depth or course error. AUV speed is chosen by the autopilot and is limited by battery charge or the onset of cavitation. Acceleration is fixed at 1 kn/s^2 , while depth and azimuth rates depend on a combination of speed and control surface angle. The vehicle's pitch angle is assigned a steady-state value determined by the AUV's speed and sternplane angle.

Although the AUV displays rigid body behavior and inertial delay, no attempt was made to model actual submarine dynamics, since these would have little impact on the large-scale decisions implemented by the mission controller. This model is a simple and effective way to display the actions and results generated by mission execution algorithms.

2) *NPS AUV-Sim2*: The second simulation system is based on the Swimmer Delivery Vehicle's dynamics and preliminary NPS model hydrodynamic test data [5]. The hull shape is a flattened cylinder with a rounded bow and a tapered stern; the AUV maneuvers with bow planes, stern planes, twin rudders, and twin screws. The dynamics model uses a vehicle mass of 12 000 lbs at neutral buoyancy with a length of 17 ft, a beam of 5 ft, and a height of 2.5 ft (the visual representation has been scaled proportionately to a length of 5 ft). The NPS Model 2 AUV (still under construction) is being equipped with two vertical and two horizontal thrusters; the simulator image shows these thrusters (see Figs. 1 and 2 for a simulator view of NPS AUV Model 2).

The AUV's position, orientation, and velocity are determined by calculating hydrodynamic drag forces and Euler angle rates and then updating these parameters at a 30-Hz rate. The AUV is displayed from an external point of view instead of the earlier "through the periscope" perspective. The simulation program is a considerable improvement over the NPS AUV Sim1 model, since the AUV exhibits realistic acceleration and inertial behavior.

NPS AUV-Sim1 [6] was designed to evaluate AUV hydrodynamic coefficients and examine the resulting vehicle dynamics under a variety of speeds and pitch angles. The simulator relies on mouse-indicated manipulation of the vehicle's control surfaces and its speed; there is no autopilot controller.

The NPS AUV-Sim2 system uses a simple autopilot depth or course-error calculation to set control surface positions for maneu-

vers. Autopilot orders create control surface angles that in turn act on the AUV hydrodynamic model to generate changes in depth or course. Although this first-order controller produces abrupt and nonlinear control surface behavior, the NPS Model 2 AUV design team is developing a more advanced control system. The simulator's modular code structure allows the advanced controller algorithm to be easily installed between the autopilot and hydrodynamic model for testing and analysis.

B. Environment

The ocean environment for NPS AUV-Sim1 is described in detail in [3]. The simulation begins with the AUV at periscope depth in a sector of water 5 nmi on a side. The seafloor of this model is a submerged cone with an exposed island (the cone's vertex) near the center of the sector. In addition to the island and its shoals, the AUV must also contend with a number of surface contacts—military vessels, merchant ships, and buoys. The large body of water and its congested environment provide a realistic test of the AUV's mission control and guidance subsystems.

The water environment for NPS AUV-Sim2 is modeled after a proposed test site for the actual vehicle. The simulation displays a "swimming pool" (120 ft by 60 ft by 8 ft deep) containing a number of submerged cylindrical obstacles. The simulator's AUV and test pool are scaled to the actual sizes of the NPS AUV and its test site, although the hydrodynamic model is not yet scaled to this environment. The test pool is a much simpler environment than the ocean environment of NPS AUV-Sim1 and is intended to give the AUV design team a realistic way to test various algorithms for mission control, guidance, and vehicle control before integrating the software with the actual AUV.

C. Missions

The simulator's mission control software is divided into four main categories: Charting, reconnaissance, surveillance, and covert payload delivery. In each category, the mission controller executes the algorithms required to maneuver the AUV to the desired location, perform its required tasks, and return the vehicle to its starting position. Additional algorithms handle other tasks or emergencies such as path planning, uncharted shallow water, or close contacts.

Lesser "missions" test the NPS Model 2 AUV's guidance and control subsystems. These tasks, subsets of the larger missions, analyze the vehicle's ability to transit and navigate in the test pool.

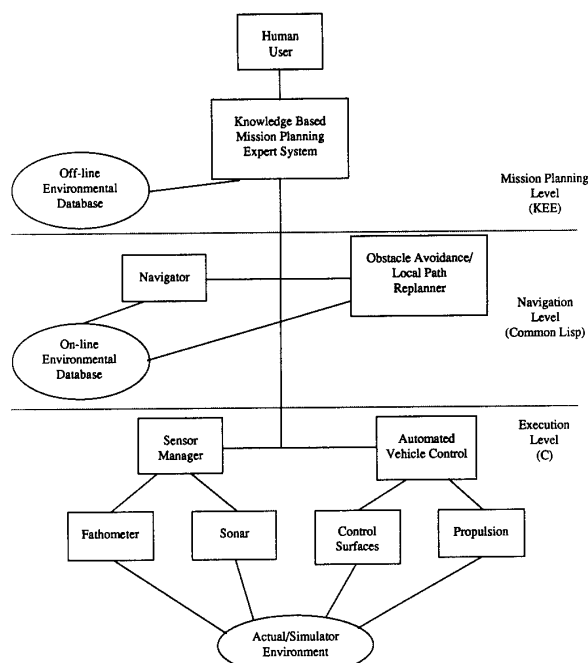


Fig. 3. Control system architecture.

This starts with simple maneuvers such as crossing the pool or circumnavigating it and builds into more complicated sequences requiring the vehicle to execute depth changes, to pass through and/or hover at specific coordinates, and to maneuver for collision avoidance.

D. Control System Architecture and Languages

The overall software architecture implemented for the NPS Model 2 AUV is shown in Fig. 3. The top level is the Mission Planning level, which is implemented with rules and a knowledge-base using the KEE (Knowledge Engineering Environment) software development shell. The user interacts directly with this level by selecting a mission and then supplying additional information via prompts. Once this information has been acquired, this level automatically plans a mission, which includes a detailed path to a goal. As a path is planned, all the known obstacles and environment data are considered. The mission is passed down to the lower level, the Navigation level, by downloading the details mission.

The navigation level is responsible for guiding the AUV following the planned path in the downloaded mission. It consists of two modules: Navigator and Obstacle Avoidance/Local Path Replanner. Both modules are written in Common LISP. The Navigator module is responsible for driving the vehicle under normal conditions. If the AUV movement is blocked by an unexpected (uncharted) obstacle, then the Obstacle Avoidance module becomes active and controls the AUV until it avoids the obstacle. If an obstacle is large enough to make the AUV deviate from the planned path, then the Local Path Replanner executes in order to return the AUV to the originally planned path.

In the Execution Level, maneuvering parameters given by the Navigation level are interpreted by the autopilot as control surface commands for the AUV's course, speed, and depth. The sensor modules get electronic and acoustic inputs and pass them to the vehicle controller or back up the hierarchy to the Navigator and the Obstacle Avoidance module, where the data is analyzed and acted on. The lowest level is written in C for fast execution speed and portability.

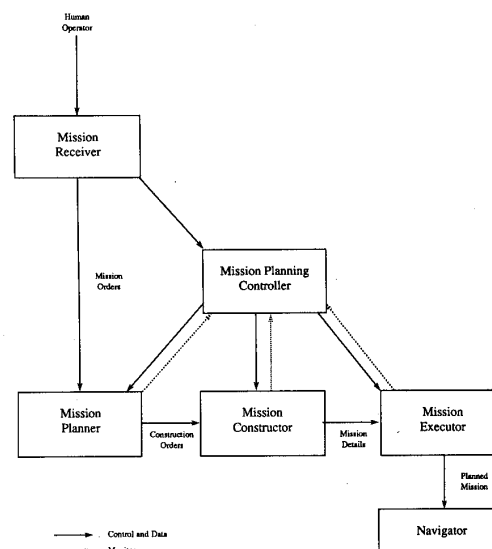


Fig. 4. Structure of Mission Planning Expert System.

The Execution level software is fully implemented in the AUV simulator, but the Navigation Level is not yet completed. Currently, only the simple functionality of the Navigator is running in the simulation environment. Online obstacle avoidance is not available.

1) *The Mission Planning Level:* The KEE software development shell organizes the Mission Planning level. This powerful software development tool runs on TI Explorer or Symbolics LISP machines. KEE provides integrated software paradigms: Rule-based and object-oriented. The former paradigm is valuable in expressing expertise demonstrated by a human, while the latter is suitable for implementing functional entities having well-defined procedural knowledge.

The Mission Planning Expert System, part of the Mission Planning Level, is composed of one supervisor called the Mission Planning Controller and three active agents: Mission Planner, Mission Constructor, and Mission Executor. The interaction among them is shown in Fig. 4. When the Mission Planning Expert System is started, the user first interacts with the Mission Receiver through parameter value panels presented on the LISP machine's monitor. After the user has specified the mission parameters to the Mission Receiver, the Mission Receiver checks the completeness of the parameters. If the user input is complete, the Mission Receiver generates the Mission Orders and initiates the operation of the Mission Planning Controller.

The Mission Planning Controller instructs the Mission Planner to work on the Mission Orders. The Mission Planning Controller monitors the operation of the Mission Planner. When the operation is finished, the Mission Planning Controller initiates the operation of the Mission Constructor. With construction orders, the Mission Constructor generates the Mission Details, which include the path description with way points and speeds along the path. When this operation is finished, the Mission Planning Controller initiates the operation of the Mission Executor. The Mission Executor downloads the Mission Details to the Navigator in the second level of the system architecture.

The Mission Planner is implemented as a rule-based system. It converts the human-oriented, unstructured information in the Mission Orders to structured and concrete information output as construction orders. The Mission Planner's major task is to analyze the Mission Orders and choose the correct search tool from the three search tools stored in and used by the Mission Constructor. Although the Mission

Planner does not possess the search tools, it knows their characteristics. Based on its knowledge about the tools, the Mission Planner examines the advantages and disadvantages of each tool and chooses the most appropriate. The Mission Planner's functionality is very close to that of a group of voters casting ballots. To simulate this functionality, the Mission Planner internally has voting rules. Each voting rule becomes a voter and gives favor values to the available path planning tools. The Mission Planner then chooses the tool that receives the highest favor values from the voting rules and includes that tool in the construction orders. Each voting rule only concerns its own specialized area. For example, a rule that is tailored to the planning time requirement produces higher values for the tools based on the comparison between the expected planning time and the given planning time. Depending on the search tool (method), the time required to plan a path in the Mission Constructor differs greatly. The user limits the planning time of the system. Thus a tool becomes more or less favorable to the system than others. Besides tool recommendation, the Mission Planner also processes environmental data in the Mission Orders and produces information usable by the Mission Constructor. For example, if the Mission Orders indicate that the surface threat is high, then the Mission Planner interprets this and generates more meaningful information to the Mission Constructor; i.e., "a shallower area than the mission depth is not allowed during path construction." The detailed implementation is reported in [7].

The Mission Constructor works on complete and well-structured input and output: The construction orders and mission details. The Mission Constructor's task is also clearly defined. Thus its functionality is implemented with a method (a procedure) and is realized with a KEE unit. It has three search tools: A*, Best-First and a Heuristic 3D, grid-based graphics search. The heuristic search method, adapted from [8] and modified for the AUV application, is included to increase the system performance, because the Heuristic search has large time and space advantages over the other search methods. The path obtained from the Heuristic search is slightly inferior to that of A* and is almost equivalent to that of Best-First for most cases. Some details concerning the Heuristic search method are reported in [8]. With the search method selected by the Mission Planner, the Mission Constructor plans a path from the start to the goal. The path consists of a series of way points to the goal. Finally, the Mission Constructor generates the Mission Details with the way points and the desired speed along the path.

The Mission Executor is implemented with a KEE unit for the same reason as that of the Mission Constructor. The Mission Executor's functionality is simple. It converts the Mission Details into a form downloadable to the Navigator in the Navigation Level.

2) *Navigation Level:* In the simulation environment, the Navigator resides in the Symbolics LISP machine, while in an actual environment this level is in the AUV vehicle mission control computer. The Navigator in the LISP machine drives the vehicle following the way points downloaded from the Mission Planning Level. Because the simulator is running on a Silicon Graphics IRIS workstation, there are frequent data exchanges via the communications interface between the LISP machine and workstation. The Navigator sends course, speed, and depth commands to the IRIS workstation and monitors the position of the AUV to ensure that the AUV follows the desired path. The Navigator continuously compares the current AUV position and desired subgoal, one of the way points along the path. The Navigator constantly corrects the vehicle movement by using a line-of-sight guidance law. The navigator converts a position error between the current position and subgoal position to velocity commands, and then feeds them to the AUV simulator. If the AUV gets within a certain distance to a subgoal, then

the Navigator chooses the next way point and uses it as a new subgoal to drive the AUV simulator. When the AUV reaches the goal, the Navigator makes the AUV return to base. To return to base, the Navigator uses the stored way points in reverse order.

3) *The Execution Level:* The Execution level is written in C and runs on the IRIS 4D/70GT graphics workstation. This level is the lowest level of AUV control; it executes either manual or autopilot commands to update vehicle and environmental displays. In autopilot mode, the Execution level receives planning-level commands for the location of the next mission subgoal, AUV course/speed/depth, and the mission phase. The Execution level code interprets these commands, positions each control surface to achieve the AUV's commands, and updates the three-dimensional visual display to show the vehicle's current orientation.

At each update of the three-dimensional visual display, the Execution level passes sensor information up to the Mission Planning level. This data is processed and can be used to alter the next set of guidance commands. An example of this occurs when the AUV's sensor reports "uncharted" shallow water or obstacles (features unknown to the navigator's environmental database), causing the planning level to alter its commands, reposition the AUV, and prevent a collision.

E. Communications Software

The execution-level code on each IRIS graphics workstation requires communications support for data exchanges with the planning-level code on the LISP machine. Both communications modules link a graphics workstation with a LISP machine via an Ethernet cable; each module passes the same data types and structures in slightly different formats. The operator selects the machine on which the simulation will be run; this determines which portions of the communications modules will be used to support the simulation. The information exchange between a LISP machine and an IRIS workstation allows the planning level to send commands to control the Execution level; the Execution level uses the communications code to send simulated sensor data back to the LISP machine for analysis. This information exchange executes in a loop that occurs about every 3 s. After carrying out its initial mission commands, the Execution level passes to the planning level data containing the AUV's present course, speed, and depth, and the depth under the keel, and sonar contact bearing/range information. The LISP machine analyzes this data and sends back the mission phase command, the coordinates of the next subgoal, and the autopilot course, speed, and depth required to reach the subgoal.

This communication code is not critical to the success of the NPS AUV and will not be used for actual AUV operations. This being the case, emphasis was placed on implementing a functional solution instead of a robust efficient subsystem. The actual NPS AUV mission control computer uses a single processor which allows data to be passed between the planning and execution level in a much quicker and more reliable way.

III. CONCLUSIONS/LIMITATIONS/PERFORMANCE

NPS AUV-Sim1 offers an excellent facility for prototyping new mission plans designed for open-ocean environments. In conjunction with the graphical simulation, researchers at NPS have developed nine different missions in this format, falling within the general areas of charting, reconnaissance, surveillance, and covert payload delivery. To support this research, a mission template concept has been adopted as an aid to human understanding and programming complex missions [3]. To create a proposed generic mission, a programmer

AUV Mission Template

Name:
 Purpose:
 Duration:
 Transit Depth:
 Transit Speed:
 On Station Time:
 Action On Station:

Mission Format:

Fig. 5. AUV mission template.

first fills out the blanks in the template in the mission template form (see Fig. 5). This can then be used as a specification to write the program for the intended mission. This concept has proved to be a valuable tool for writing mission and planning-level LISP code.

NSP AUV-Sim2 is an important tool for incorporating new autonomous control concepts and algorithms into the latest version of the NPS AUV. The KEE expert system shell provides an inexperienced operator with an interactive (user-prompt and mouse-driven) Mission Planning Control Panel structure for rapid mission planning and execution. The Mission Planning Control Panel is a prototype for control panels that may be placed on actual AUV deployment platforms in the future. The shell also provides a powerful environment in which programmers can modify the simulator's mission-level code and develop additional missions [4]. The faster and more powerful IRIS 4D/70GT graphics workstation effectively simulates the AUV's actual operation with a real-time display of the vehicle's actions, and this workstation has the capacity to accommodate more complex AUV models or controllers.

The simulator is a valuable test and debugging environment which will save countless hours of experimentation; it will also verify code reliability before the software is installed in the actual NPS Model 2 AUV computer system. NPS AUV-Sim2 provides the operator with a wide choice of starting locations and viewing positions to thoroughly examine vehicle performance from many different perspectives. This viewing flexibility greatly reduces the risks, simplifies the logistics, and minimizes the costs of testing the NPS Model 2 AUV in its ocean environment.

REFERENCES

- [1] R. C. Robinson, "National defense applications of autonomous underwater vehicles," *IEEE J. Oceanic Eng.*, vol. OE-11, pp. 462-467, Oct. 1986.
- [2] S. Eisenstadt, "Navy envisions \$5 billion ASW minisub fleet," *Defense News*, vol. 2, no. 51, p. 1, Dec. 24, 1987.
- [3] D. L. MacPherson, "A computer simulation study of rule-based control of an autonomous underwater vehicle," Master's thesis, Naval Postgraduate School, Monterey, CA, June 1988.
- [4] D. B. Nordman, "A computer simulation study of mission planning and control for the NPS autonomous underwater vehicle," Master's thesis, Naval Postgraduate School, Monterey, CA, June 1989.
- [5] G. L. MacDonald, "Model-based compensator design and experimental verification of control systems for a model AUV," Master's thesis, Naval Postgraduate School, Monterey, CA, Mar. 1989.
- [6] M. Schwartz, "Systems identification and control for an AUV," Master's thesis, Naval Postgraduate School, Monterey, CA, Mar. 1989.
- [7] S. H. Kwak, S. M. Ong, and R. B. McGhee, "A mission planning expert system for autonomous underwater vehicles," in *Proc. Symp. Autonomous Underwater Vehicle Techn.* (Washington, DC), June 6, 1990, to be published.
- [8] D. K. Ok, "A computer simulation study of a sensor-based heuristic navigation for three dimensional rough terrain with obstacles," Master's thesis, Naval Postgraduate School, Monterey, CA, June 1989.

Using Common LISP in the EAVE Autonomous Underwater Vehicle

PAUL S. BOWEN, STEVEN G. CHAPPELL, AND ROGER GONZALEZ

Abstract—The Marine Systems Engineering Laboratory of the University of New Hampshire has ported the University of Utah's Portable Common LISP Subset (PLCS) to the EAVE underwater autonomous vehicle. The use of Common LISP in the EAVE autonomous vehicle is expected to improve programmer productivity and software portability. Also, the use of the LISP interpreter will allow for software changes to be made while in the field, thus saving time during vehicle operations.

Issues concerning the operation of LISP in a real-time environment, such as the impact of garbage collection, have been resolved by using an efficient version of Common LISP and by using LISP at the high-end of a time-based software hierarchy.

Keywords—Autonomous vehicles, PCLS, porting, pSOS, embedded systems.

I. INTRODUCTION

The Marine Systems Engineering Laboratory (MSEL) at the University of New Hampshire has been involved in the design, development, and operation of several generations of autonomous underwater vehicles [1]–[3]. The first vehicle developed by MSEL, the Experimental Autonomous Vehicle (EAVE), was designed to perform pre-programmed tasks without any human intervention once the mission was underway. The computing system was designed and developed in-house and consisted of one Motorola M68000 and three Harris 6100 processors with a maximum of 32 kilobytes each of RAM. This sufficed to control the vehicle and provide minimal navigation and sensor management.

The desire for increased capability and flexibility led to the development of the second- and third-generation vehicles. Today, EAVE's computing hardware consists of two levels of computing systems: A lower level consisting of three Motorola M68000 processors, and an upper level which is made up of several VME-based M68020 processors, each with 1–4 Megabytes of RAM. Both systems run the Software Components Group's pSOS, which is a small and efficient real-time operating system. The lower-level computers operate the thrusters and interface directly with the sensors. The high-level computers provide a platform for the more advanced control algorithms such as situation assessment, mission planning, and world models.

The tremendous gains in hardware capacity can be contrasted with a lag in the development of the vehicle's high-level software, thus delaying the realization of the full potential of the current system. The primary reason for this lag is due to the difficulty of developing experimental software in a conventional programming environment. In our estimation, one solution to the software development bottleneck is to introduce a programming environment which facilitates the development of sophisticated software, offers device independence, and improves on the ability to diagnose and effect changes in the field.

Towards that end, MSEL has ported the University of Utah's Portable Common LISP Subset (PCLS) to our UNIX-based software

Manuscript received October 1989; revised March 27, 1990. This work was partially supported by the National Science Foundation through Grant EID-8818406.

The authors are with the Marine System Engineering Laboratory, University of New Hampshire, Durham, NH 03824.

IEEE Log Number 9036196.