NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

R68555

A STUDY OF 3-D VISUALIZATION
AND KNOWLEDGE-BASED
MISSION PLANNING AND CONTROL
FOR THE NPS MODEL 2
AUTONOMOUS UNDERWATER VEHICLE

by

Ray Charles Rogers

December 1989

Thesis Advisor: Professor Robert B. McGhee

Approved for public release; distribution is unlimited.

T247304

Unclassified
Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report Approved for public release; distribution is unlimited.
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization Naval Postgraduate School | 6b Office Symbol Code52Mz	7a Name of Monitoring Organization Naval Surface Warfare Center
6c Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	7b Address *(city, state, and ZIP code)* White Oak, MD 20910
8a Name of Funding/Sponsoring Organization Naval Postgraduate School | 8b Office Symbol Code 52Mz	9 Procurement Instrument Identification Number O & MN, Direct Funding
8c Address *(city, state, and ZIP code)* Monterey, CA 93934-5000	10 Source of Funding Numbers

Program Element Number	Project No	Task No	Work Unit Accession No

11 Title *(Include Security Classification)* A STUDY OF 3-D VISUALIZATION AND KNOWLEDGE-BASED MISSION PLANNING AND CONTROL FOR THE NPS MODEL 2 AUTONOMOUS UNDERWATER VEHICLE

12 Personal Author(s) ROGERS, Ray C.

13a Type of Report	13b Time Covered	14 Date of Report *(year, month,day)*	15 Page Count
Master's Thesis	From To	December 1989	115

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Field	Group	Subgroup	Autonomous underwater vehicles, artificial intelligence, robotics, 3-D computer graphics

19 Abstract *(continue on reverse if necessary and identify by block number*

Recently, specific tasking/total military mission concepts for subsea tasks have been developed that demand either substantially more endurance and range than can be provided by manned submersibles and Remotely Operated Vehicles (ROVs), respectively. Small, autonomous unmanned systems can provide the best combination of speed, endurance, range, depth capacity, and flexibility needed to make these concepts realizable.

As the U.S. military has continued to identify more and more tasks that can be performed by autonomous systems, the Naval Postgraduate School has heightened its research efforts to develop an experimental autonomous underwater vehicle (AUV) to address these military requirements. As part of this development process, a series of NPS AUV simulation systems have been developed that couple knowledge-based mission planning and control systems with 3-D visualization (graphics) workstations that communicate across an interprocess communications network. Development of these simulation systems have produced an extremely useful "laboratory environment" for rapid prototyping of AUV planning, navigation, and control subsystems software. This thesis updates and extends the formats and functionality of the simulation systems to include a Mission Planning and Control Workstation as a prototype for use aboard AUV deployment vessels, upgrades mission profiles, and incorporates improvements to the software interface between the mission planning and control subsystem and the 3-D visualization subsystem.

20 Distribution/Availability of Abstract [✓ unclassified/unlimited] same as report DTIC users	21 Abstract Security Classification Unclassified	
22a Name of Responsible Individual Robert B. McGhee	22b Telephone *(Include Area code)* (408) 646-2449	22c Office Symbol Code 52Mz

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page
Unclassified

A STUDY OF 3-D VISUALIZATION AND KNOWLEDGE-BASED MISSION PLANNING AND CONTROL FOR THE NPS MODEL 2 AUTONOMOUS UNDERWATER VEHICLE

by

Ray Charles Rogers
Lieutenant Commander, United States Navy
B.S., Southern University and A & M College, 1978

Submitted in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

ABSTRACT

Recently, specific tasking/total military mission concepts for subsea tasks have been developed that demand substantially more endurance and range than can be provided by manned submersibles and Remotely Operated Vehicles (ROVs), respectively. Small, autonomous unmanned systems can provide the best combination of speed, endurance, range, depth capacity, and flexibility needed to make these concepts realizable.

As the U.S. military has continued to identify more and more tasks that can be performed by autonomous systems, the Naval Postgraduate School has heightened its research efforts to develop an experimental autonomous underwater vehicle (AUV) to address these military requirements. As part of this development process, a series of NPS AUV simulation systems have been developed that couple knowledge-based mission planning and control systems with 3-D visualization (graphics) workstations that communicate across an interprocess communications network. Development of these simulation systems have produced an extremely useful "laboratory environment" for rapid prototyping of AUV planning, navigation, and control subsystems software. This thesis updates and extends the formats and functionality of the simulation systems to include a Mission Planning and Control Workstation as a prototype for use aboard AUV deployment vessels, upgrades mission profiles, and incorporates improvements to the software interface between the mission planning and control subsystem and the 3-D visualization subsystem.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The original code for the NPS AUV-SIM1 simulation system was provided by Dave MacPherson and was modified by Doug Nordman (NPS AUV-SIM2) to conform to new hardware and different operating systems. In this thesis Nordman's work has been further modified to extend the use of the KEE software development shell by developing an interactive and user-friendly Mission Planning and Control Workstation. The original NPS AUV dynamic simulator was developed during a graphics project by Dave Marco, Ray Rogers and Mike Schwartz. The dynamic simulator was later modified by Dave Marco to include an enhanced drawing model and a more robust controller. This model has been further modified to integrate it into a full simulation system (NPS AUV-SIM3). The network communications software was not modified during this research and was written by Ted Barrow, Sehung Kwak, Bill Teter, and Larry Shannon.

Professor Sehung Kwak offered seemingly infinite wisdom and knowledge of all facets of using and developing systems on the LISP machines and their supporting software systems. His help was invaluable getting me started on the right path.

I am especially grateful to Professor Robert McGhee for allowing me the opportunity to work with him on this exciting project.

And lastly, I must thank my wife and family for allowing me the freedom of movement, without hassles, to get the job done.

I. INTRODUCTION

A. BACKGROUND AND BRIEF PROBLEM STATEMENT

Research in the area of autonomous vehicles has been extensively applied to ground, undersea, and airborne systems with varying degrees of success over the last three decades. This thesis focuses in part, on the uneven evolution of Autonomous Underwater Vehicles (AUVs) and the concurrent developmental trends of the expert systems with which they are controlled.

Underwater vehicles are either tethered or autonomous, manned or unmanned. Tethered systems, whether manned or unmanned, have the advantage of unlimited power supplied by a surface support platform, but are severely restricted in their range and ability to maneuver due to their support cabling [Ref. 1:p. 33]. Untethered manned systems, such as the Alvin, Johnson-Sea Link, and Deep Rover [Ref. 1:p.33], avoid the problems inherent to tether management, but are nontheless restricted to relatively shallow depths, slow speeds, and short endurance. Small, autonomous unmanned systems, however, can avoid all of the problems listed above and provide the combination of speed, endurance, depth capacity, and flexibility needed for today's sophisticated military applications and marine industrial pursuits.

Interest in Autonomous Underwater Vehicles on the part of the military and industry in general began in the early 1960s and conceptual designs began to evolve by the middle of that decade [Ref. 2:p. 60]. This interest quickly peaked, however, and dwindled back to a low level for many years to come. First, manned submersibles captured the imagination of the military and deep-sea related industry from about 1965 to 1975, but was followed immediately (and somewhat overlapped) by Remotely Operated Vehicles (ROVs) from roughly the 1970's to the recent past.

During this era, almost all the required subsea tasks could be accomplished by manned submersibles (with their limited endurance) or by ROVs (with their limited range) [Ref. 2:p. 61]. It was not until specific tasking/total mission concepts were developed demanding substantially more endurance and range that AUVs came to the forefront of general interest again. Simultaneously, the escalating cost of manned systems coupled with the rapid technological advances of traditional military adversaries, created an atmosphere to seriously consider AUVs for multi-purpose, multi-sensor military applications.

A fair number of autonomous systems that are operational today are controlled in general by some form of artificial intelligence (AI) program, and in particular, by an Expert System, which is a subfield of AI that has found wide practicality for use in industrial and naval engineering applications. The developmental trends of AI have been almost as cyclic as that of the autonomous underwater vehicle. For several decades, researchers have dreamed of "...autonomous 'thinking' machines that are free of human control" [Ref. 3:p. 32]. And now some believe we are not far from realizing that dream. Daniels [Ref. 4:p. 23] defines AI as follows: "AI is defined as the application of knowledge, thought and learning to computer systems to aid humans." No matter what the real definition is, the important aspect of AI is that this is a relatively new field of computer implementation which manipulates knowledge and symbols in ways that are not possible with conventional data processing.

AI got its rather auspicious start in 1956 when the phrase was coined by John McCarthy, the inventor of the LISP programming language [Refs. 5, 6, 7]. Since then, AI has had at least two periods of heightened expectations and crushed hopes. During the late 1950s and early 1960s researchers and scientists focused on autonomous systems and the 1960s saw a number of implementations tested. For the most part these attempts ended in failure [Ref. 5]. The situation has changed dramatically over the past seven years. Generic knowledge systems which embody natural language interfaces, tools for developing the expert knowledge base through rules and examples, and inference engines are now commercially available for desk top microcomputers [Ref. 3:p. 34].

As the U. S. military has continued to identify more and more tasks that can be performed by autonomous systems, the Naval Postgraduate School (NPS) has heightened its research in 3-D visualization techniques, knowledge-based expert systems, and development of an experimental AUV to address these military requirements. Previous 3-D visualization and expert systems research at NPS has shown the outstanding utility of coupling knowledge-based mission planning and control systems with 3-D graphics workstations to produce an extremely useful "laboratory environment" for testing AUV planning, navigation, and control subsystems [Refs. 8, 9]. Use of these visual simulators have significantly reduced the time and expense of implementing various AUV subsystems while also permitting efforts to proceed along several independent but simultaneous approaches.

This thesis improves on previous research at NPS by further expanding and upgrading the missions and mission planning software and improving the software interface between the mission planning and control subsystem and the 3-D visualization subsystem. In addition, an interactive AUV Mission Planning and Control Workstation was developed as an experimental prototype for possible use onboard AUV deployment platforms. This workstation has been fully integrated with the AUV mission planner and graphics workstation to more fully demonstrate the total AUV environment.

B. THESIS ORGANIZATION

Chapter II reviews previous work on AUV systems and examines the development of expert systems and their significant role in the evolution of AUVs. Expected developments in the area of expert systems and their possible impact upon future AUV research are also discussed.

Chapter III presents a detailed problem statement for this research and describes the evolutionary development of the NPS AUV, to include a comparison of the NPS Model 1 and NPS Model 2 AUVs. The mission upgrades and mission planner/dynamic model interface upgrades are discussed and contrasted to earlier versions. Lastly, the AUV Mission Planning and Control Workstation is described and the utility of this prototype for use in the fleet is discussed.

A detailed description of the simulator's operation is described and discussed in Chapter IV. This includes a review of the mission planning and control software architecture and how they relate to the two most recently developed 3-D visual simulators, AUV-SIM2 and AUV-SIM3, the operation of the AUV Mission Planning and Control Workstation, and a user's manual.

The utility of the AUV simulator research is examined in Chapter V. This chapter explains how rapid prototyping using the expert system control software and 3-D visualization system has saved time and cost in the development of the NPS AUV. These results are summarized in Chapter VI and are used as a basis for proposed research extensions. This chapter also contrasts the development of the NPS AUV and systems developed by other research organizations.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

The 1980s have witnessed a virtual boom in Autonomous Underwater Vehicle (AUV) technological advances as compared to the uneven interest and development of the previous three decades. Recent advances in micro-electronic technology, high speed digital computers, component miniaturization, artificially intelligent signal processing, control and sensor systems, and high-energy, high-density power supplies, coupled with rapidly broadening industrial and military demands for this technology, has made this revival both necessary and possible. MacPherson and Nordman [Refs. 8, 9] reported on recent developments in AUV technology used for military and industrial applications. Research and development of even more innovative and versatile systems continue.

As far back as the early 1970s, the U. S. Navy recognized the rapidly growing requirement for deep-ocean unmanned vehicles to perform survey, recovery, and other classified military missions[Ref. 2:p. 62]. Traditional adversaries were becoming more technologically advanced at an alarming rate which created a need for more innovative counters to this very real threat. This type of vehicle outfitted with appropriate sensors and highly efficient computerized decision and control technology offered rapid response to emergency situations while, at the same time, providing an economical and safe means of fulfilling a broad spectrum of tasks.

Now, nearly twenty years later, the requirements have not changed but have grown increasingly more complex [Ref. 2:p. 65].

To meet these complex demands for AUV technology, expert system technology, as a branch of Artificial Intelligence (AI), has had to follow suit. Most autonomous systems use some form of knowledge-based expert system to manage their computer software hierarchy/control structure. After experiencing relatively auspicious beginnings, much like that of the AUV, the ever-increasing computer capacity, significant breakthroughs in knowledge representation, and the high efficiency of symbolic programming techniques, has enabled expert systems to become more and more popular in a wide variety of application fields where highly reliable and rapid decision-making is involved, such as medical diagnosis, geographical pattern search, and even in monitoring complex process control systems [Ref. 10:p. 165].

This chapter examines current AUV and expert systems advances and then shows how these systems relate to today's myriad and complex industrial and military demands.

B. AUTONOMOUS UNDERWATER VEHICLE TECHNOLOGY

Over the last 20 years, the trend in undersea vehicles has progressed from manned submersibles to Remotely Operated Vehicles (ROVs) to Autonomous Underwater Vehicles (AUVs). This progression has been directed toward minimizing the need of man's physical presence and intervention underwater by developing a self-contained, preprogrammed, decision-making AUV, that is independent of all external control, with the exception of launch and recovery operations, and in some cases, midcourse data recovery or redirection [Ref. 11]. This research and development effort extends to Multiple Autonomous Underwater Vehicles (MAUVs) and AUV-ROV combinations (i.e., the ALVIN submersible and JASON JR (ROV) used during the TITANTIC exploration).

The greatest advantage of an AUV is that it can swim free from restrictions of umbilical cabling and is capable of operating at substantial depths with relatively long endurance [Ref. 12:p. 263]. AUVs with subsea robotics capability provide a great challenge and opportunity for automating many of the present underwater vehicle applications currently performed by manned or remotely operated submersibles. Subsea robotics capabilities needed by AUVs to accomplish these tasks include advances in areas such as: high resolution, 3-D imaging systems; computer aided vision systems; satellite-subsurface acoustic/laser telemetry systems; highly dextrous, autonomous two or three arm manipulator systems; and high-energy, high-density power sources [Ref 11]. A myriad of research and development efforts in these areas is in progress both in the United States and abroad and this section surveys some of those activities; in particular, those activities sponsored by the Department of Defense.

1. R & D Background

A recent survey revealed that 36 different U.S. organizations are conducting research/development projects involving underwater vehicles, 26 of which are directly related to AUVs. In addition, 10 different foreign countries are involved in similar R&D efforts. In the U.S., the major source of funding for AUV R&D is the Department of Defense, which accounts for approximately 90% or more of the total funds. Within the DOD, the Navy and the Defense Advance Research Agency (DARPA) are the primary backers of the subject research. The survey also reports that non-DOD efforts account for less than about $500,000 per year [Ref. 11].

2. DOD AUV Research Programs

a. Draper Laboratories

Working under the auspices of DARPA, Draper Laboratories is developing the Unmanned Underwater Vehicle (UUV). The UUV shape will be that of a submarine, the physical characteristics of which is consistent with the low drag needed for minimum propulsion power. Other characteristics of the UUV are listed in Figure 2.1 [Ref. 11].

DRAPER LABORATORIES' UUV CHARACTERISTICS

*Shape:	submarine
*Weight:	approximately 6,800 kg
*Speed:	10 knots (maximum)
*Acceleration:	0 to 10 knots in 44 secs
*Depth Control:	+/- 1m at speeds greater than 3 knots
*Navigational Accuracy:	accurate to within 0.2 knots
*Power:	silver-zinc battery (~2,300 kg)

Figure 2.1 Draper Laboratories' UUV Characteristics

b. Martin Marietta

Also working under the DARPA umbrella, Martin Marietta is involved in classified research to assess the capabilities and limitations of AUVs. Some of the tasks being considered include planting submarine sensors on the sea-floor, surveying and mapping minefields, towing hydrophone arrays and serving as long-range weapons platforms [Ref. 11].

c. Office of Naval Research (ONR)

ONR is funding, through the Naval Research Laboratory, development of several AUV capabilities, including long-range navigation and fuel-cell power sources. The long-range navigation effort is to develop a technique that will provide accurate navigational capabilities beyond 100 nautical miles from a starting point. The goal of the fuel-cell program is to develop a power source for a small unmanned observation vehicle (UOV) that will use proton exchange technology to develop a fuel cell that will not exceed the space occupied by silver-zinc batteries currently used on AUVs [Ref. 11].

d. Naval Ocean Systems Center (NOSC)

NOSC San Diego, CA has been a pioneer in the field of ROV technology since the 1960s and began AUV activity in the 1970s with development of the Experimental Autonomous Vehicle-West (EAVE-West), followed by the Advanced Unmanned Search System (AUSS) in the 1980s. The AUSS is a deep diving submersible designed for depths in excess of 6,000 meters.

Currently, the next generation AUSS is being built and is scheduled to go back to sea in 1990, and the EAVE-West is being configured for evaluating new technologies for energy sources, propulsion, data storage, machine vision, and a new processor that uses multi-computing techniques (transputers) [Ref. 13]. Additionally, another AUV recently developed by NOSC, is the Free Swimming Mine Neutralization Vehicle (FSMNV). This vehicle and the EAVE-West are being outfitted with the new processor and will be used as test beds to evaluate its performance. The new control system consists of a 16-node array of transputers hosted by an IBM-AT compatible computer and installed in an undersea electronics bottle. This test-bed processor is expected to provide valuable insights concerning undersea application of embeddable multi-computing [Ref. 14 and 15].

e. Office of Naval Technology (ONT)

In 1987, ONT funded Texas A&M University via the Naval System Warfare Center (NSWC) for AUV research and development. The current work is directed at developing redundant fault tolerant control systems for long duration missions for AUVs [Ref. 11].

f. National Oceanic & Atmospheric Administration (NOAA)

The Sea Grant Office of NOAA has funded MIT to develop a lightweight, low-cost autonomous vehicle called SEA SQUIRT. The vehicle is intended to be a platform for testing AI algorithms. A layered control approach to AI is used to give the vehicle the ability to respond to unanticipated circumstances and environments. The vehicle will be capable of obstacle avoidance and sensor mapping of objects of interest [Ref. 11].

g. National Science Foundation (NSF)

NSF has sponsored AUV development projects through four different institutions: University of New Hampshire, Florida Atlantic University, Carnegie-Mellon University, and Woods Hole Oceanographic Institution. Research conducted at the University of New Hampshire pertains to the design of real-time, expert systems for autonomous vehicles. This project will also assess some of the problems in utilizing lnowledge-based systems in real-time operations [Ref. 11].

A joint research project by Florida Atlantic University and Carnegie-Mellon University is concerned with the development of an underwater, 3-D vision system for intelligent AUVs. Algorithms will be developed to process signals from a multiple element sonar array to generate a grid representation of the ocean bottom terrain, and the development of map building algorithms to generate a coherent map of the terrain. The goal of this research is to develop the capability to produce a high level, 3-D world model image required to support underwater surveying and sensor-based navigation [Ref. 11].

Woods Hole Oceanographic Institution's project involves the design and construction of an unmanned, untethered vehicle for servicing long-term deep ocean benthic experiments. This vehicle, called the *Autonomous Benthic Explorer,* can be launched from any oceanographic research ship and will remain on site for several months, during which time, it will periodically move about in its acoustic navigation net, taking photos and making a variety of scientific measurements. At the end of its mission, it can be recovered, on command, by an available ship. [Ref. 11]

3. Industrial Activities

A number of industrial activities are also involved in AUV R&D efforts, several of which are in direct support of DOD initiatives. These industrial activities include: Boeing Aerospace, Eltech Research Corporation, Martin Marietta (discussed in previous section), Westinghouse Electric Corporation R&D Division, Honeywell, INC. Systems & Research Center, and Hughes Aircraft.

Boeing Aerospace is involved in development of a long endurance power system that will be capable of providing at least 1,000 kw-hours of energy over a 10 day period [Ref. 11]. Eltech has developed an aluminum air fuel cell--an electrochemical device that continuously converts the chemical energy of aluminum and oxygen into electrical energy. The energy yield of this configuration is 200 to 300 kw-hours per pound, as compared to the silver-zinc batteries currently in use that yields about 100 kw-hours per pound [Ref. 11]. Westinghouse has developed a silver-iron battery for use in AUVs where high-energy, high-density power systems are required. Additionally, their Oceanic Division has been examining various mechnisms for extending laminar flow on underwater vehicles. [Ref. 11]

Martin Marietta, in addition to development of their UUV, is conducting research on a nonlinear control technique known as sliding mode control (SMC), and also, Intelligent Waypoint Transiting in complex AUV environs. The UUV is used as an in-water test bed for the research, and the SMC is designed to provide a theoretical framework for the design of controllers that are robust and are able to adapt to varying payloads

[Ref. 11]. Intelligent Waypoint Transiting is an approach for planning and executing waypoint transiting in a complex and dynamic ocean scene that requires the high-level controller to "learn" its environment for future use, should it be required to retraverse the same area. [Ref. 11]

Honeywell's project pertains to AUVs with complex capabilities and intelligent software for multi-mission capabilities. This project intends to greatly simplify real-time, on-scene mission programming of the AUVs on-board computer by operational personnel. A three-level approach will be developed consisting of: (1) onshore development of operational tactics: (2) predeployment programming by operational specifications, and (3) real-time, dynamic determination of the current situation and response within the desired operational constraints. [Ref. 11]

Hughes Aircraft Company, Ground Systems Group sponsors a program that envolves a multi-year effort encompassing all aspects of UUV technology. The major emphasis, however, is to develop a real-time system with intelligent planning. [Ref. 11]

C. EXPERT SYSTEMS DEVELOPMENT

1. Evolution and Characteristics

Over the last several decades, society has had an infatuation with trying to breathe life or intelligence into machines. "We no longer want computers to just add, subtract, multiply, or divide, but to act human, to think" [Ref. 16:p. 1]. One can readily imagine the endless possibilities of intelligent machines: computer systems that recommend profitable financial and marketing strategies; eagerly perform dangerous and monotonous manufacturing or exploration tasks; create new designs in the automobile or semiconductor industries; and quickly monitor and diagnose a patient's health. To satisfy this infatuation, an entirely new research effort dedicated to the development of artificial intelligence (AI) has evolved and has grown in significance to become a virtual growth industry in today's world. This new research effort is the development and widespread use of "knowledge-based expert systems." Expert systems provide the "intelligence" for the sophisticated mechanized "help-mates" of today's society.

What do people mean when they say "artificial intelligence"? What is an expert system"? Artificial intelligence research is often defined as the search for general computational models of human intelligence [Ref. 17:p. xix]. Expert systems are computer systems, comprising both hardware and software that mimic an expert's thought processes to solve complex problems in a given field [Ref. 16:p. 3]. An expert system finds reasonable solutions to problems for which there may be no hard and fast "right" answer. The "expert" computer system uses extensive experience-based knowledge of a subject to guess intelligently in the same way as a human expert.

a. Characteristics and Components

By definition, expert systems are used to solve problems or make decisions. Expert systems operate on a processing level higher than that of conventional programs. They function like a thought process--they make inferences[1] and guesses and ask questions for additional information. Suitable applications for expert systems fall into the categories listed in Figure 2.2 [Ref. 16:pp. 8-9].

CHARACTERISTICS OF EXPERT SYSTEMS

****Interpreting and Identifying:*** Explaining summarized results from input information.
****Predicting:*** Inferring likely consequences of given or hypothetical situations.
****Diagnosing:*** Identifying causes, given symptoms.
****Designing:*** Configuring objects into systems, given constraints.
****Planning:*** Devising a method for making or doing something in order to achieve an end.
****Monitoring:*** Comparing observations with established standards.
****Debugging and Testing:*** Prescribing remedies for malfunctions.
****Instructing and Training:*** Educating and transferring information.
****Controlling:*** Regulating or guiding the operation of a machine, apparatus, or system.

Figure 2.2 Characteristics of Expert Systems

The four basic components of an expert system are (1) the knowledge base, (2) the inference engine, (3) the interface, and (4) the

[1] **Inference:** An implied relationship of one object to another, allowing new facts to be derived from existing facts.

development engine. Figure 2.3 outlines the basic characteristics of each component [Ref. 16:p. 13].

COMPONENTS OF AN EXPERT SYSTEM

****Knowledge Base:*** Houses the information used by the expert system in pursuit of a solution to a problem.
****Inference Engine:*** The workhorse of the expert system. It consists of the processes that work the knowledge base, do analyses, form hypothesis, and audit the processes according to some strategy that emulates the expert's reasoning.
****Interface:*** Includes a terminal (TTY screen), graphical representations (visuals), multiple character windows, and multiple graphic windows.
****Development Engine:*** Editor or knowledge acquisition subsystem that allows the knowledge engineer to create, modify, add, and delete information from the knowledge base.

Figure 2.3 Components of an Expert System

b. Brief History

In 1965, researchers at Stanford University began work on the grandfather of all expert systems, DENDRAL [Ref. 18]. DENDRAL, based on an algorithm developed by Nobel Prize-winning chemist Joshua Lederberg, was designed to analyze information from a spectroscopic analysis on chemical compounds to determine their molecular structures. Using an efficient variant of a generate-and-test search technique in its problem solving, DENDRAL outperformed some of the best human experts in the field [Ref. 16:p. 5]. Approximately 15 years were spent in developing DENDRAL--extracting heuristic[2] information from expert chemists; formulating the experts' reasoning rules into formal rules, and implementing and testing the final system. Programmed in LISP, DENDRAL is a good example of a rule-based system, storing much of its knowledge in "If-Then" production rule statements.

In 1970, CADAUCEUS was developed at the University of Pittsburgh to aid physicians in the diagnosis of human internal diseases. Nineteen years later, this system has over 100,000 programmed relationships which represent 85% of all relevant knowledge in this particular domain. CADAUCEUS analyzes by initially examining the problem using a bottom-up problem-solving strategy and then switching to a top-down strategy, thus squeezing in on the diagnosis. [Ref. 16]

MACSYMA was written in the late 1960s by MITs Laboratory of Computer Science as a mathematical problem solving aid. By 1971 it was being successfully employed in sophisticated symbolic mathematical analysis. MACSYMA surpasses most human experts by performing differential and integral calculus symbolically and simplifying symbolic expressions. Comprising more than 300,000 lines of LISP program code, MACSYMA represents approximately 100 work-years of development time. [Ref. 16]

---

[2] **Heuristics**, in AI jargon, is a list of rules of thumb applied to a certain application or situation. They are the mainstay of the **knowledge** we try to store for use by natural English systems, expert systems, and robots.

MYCIN was developed at Stanford University in 1972 [Ref. 19]. One of the most publicized and famous expert systems, MYCIN assists with the diagnosis and treatment of infectious blood diseases. Its knowledge base currently has more than 4200 production rules. From MYCIN stemmed TIERESIAS in 1976 and EMYCIN in 1978. TIERESIAS is a knowledge acquisition tool that assists with entering and updating the MYCIN knowledge base by utilizing metaknowledge (knowledge about knowledge). EMYCIN contains all of the logical structure of MYCIN, with the exception of its knowledge of infectious blood diseases, hence the name "Empty MYCIN." Thus was born the expert system shell, a program containing logical structures and thinking strategies, but without the knowledge base of a specific domain. [Ref. 19]

Then came PUFF, a diagnostic consultation expert system for pulmonary function diseases [Ref. 19]. PUFF is a derivative of EMYCIN with a pulmonary function diseases knowledge base added.

About the same time, the Stanford Research Institute (SRI) constructed the PROSPECTOR expert system [Ref. 20]. It is a rule-based system that assists with the analysis of information related to geological exploration. Its data structure is based on a semantic network.

In 1980, XCON, developed by Digital Equipment Corporation (DEC), became the first expert system to be used successfully on a daily basis in a commercial environment [Ref. 18]. XCON performs the difficult job of configuring VAX computer systems as requested by DEC's customers. As reported by DEC, XCON saves the corporation approximately $200,000 per month in staff costs alone, not to mention savings in manufacturing costs.

The 1980s have seen a myriad of applications of expert systems that are too numerous to mention here. These new applications include autonomous system control, space systems scheduling and control, equipment/system design, traffic control, and 3-D vision systems, just to name a few. The next section sites a few examples of how expert systems are being used in the research and development of one of these areas--autonomous underwater vehicles.

2. Role of Expert Systems in AUV Development

As described in the previous section on AUV development, the success of AUV technology depends on its ability to exercise intelligent behavior in a hostile marine environment without intervention from man. The usefulness of "intelligent controllers" designed from knowledge-based expert systems for real-time operations is readily apparent. As such, many developers are concentrating their efforts on this technology.

Honeywell INCs project to develop intelligent software for AUV multi-mission capabilities uses an expert system to produce a user-friendly, simplified programming environment for operational personnel [Ref. 11]. UUV R&D efforts by Hughes Aircraft is another prime example of expert system use in the development of AUV planning and control systems. Their development of a real-time system with intelligent planning includes a graphic human interface that facilitates operator entry of mission planning data and allows quick examination of mission progress. An expert system shell is used for this facility. [Ref. 11]

Researchers at the Applied Physics Laboratory/Johns Hopkins University have developed a multiple knowledge-base approach to AUV mission control in naval applications which is based on representation of the Navy watch team model in a distributed computer system. This "human paradigm" approach was chosen in order to create a control structure that could readily accomodate knowledge from domain experts on an actual submarine [Ref. 21:p. 15]. By decomposing the control problem into a set of sub-problems that faithfully correspond to the problem domains of the members of a watch team (Captain, Conning Officer, Navigator, Engineering Officer, Communications Officer, and Helm), the problem of the knowledge acquisition bottleneck is partially addressed. The experienced watch team members become a ready source of expert knowledge. This approach, which has become a model used by other AUV developers, was tested on a network of 32-bit (Sun) Workstations. Use of expert systems allowed object-oriented design techniques to be highly modular and to be distributed on multiple processors with interprocess communications to be accomplished within and across machine boundaries. Programs which were implemented in LISP, C++, and C were able to communicate freely using this scheme. A user operating the simulation test facility is able to interface easily with the graphical interface to input maps and top level mission descriptions and then watch the dialogue between the knowledge bases as the missions are executed. [Ref. 21]

These are but a few examples of how expert systems have become an integral part of AUV development and with the current rapid developmental trends of computer, microelectronic, and processor technology, the future seems limitless.

3. Future Developments

In the past, most expert systems development efforts were very expensive and were funded primarily by government agencies as internal R&D projects since widespread commercial use was not viewed as a viable alternative [Ref 16:p. 247]. Today, both AI and its offspring, expert systems, are unquestionably growth industries with far-ranging applications in the industrial, military, and commercial sector. Researchers and developers have found that expert systems can be cost effectively implemented now and with current technological advances the future is exceedingly bright in all sectors. Figure 2.4 lists advances that are in development now that will further revolutionize the use of expert systems.

D. EXPERT SYSTEMS AND 3-D VISUALIZATION OF THE NPS AUV DYNAMICS AND CONTROL

As the U. S. military has continued to identify more and more tasks that can be performed by autonomous systems, the Naval Postgraduate School (NPS) has heightened its research in 3-D visualization techniques, knowledge-based expert systems, and development of an experimental AUV to address these military requirements. Previous research by McPherson and Nordman [Refs. 8,9] demonstrated the outstanding utility of interfacing single-user workstations that couple an expert system that coordinates mission planning and control to a computer graphics workstation which houses a 3-D dynamic model of the AUV currently under construction at NPS. As in the research being conducted at APL/Johns Hopkins University, on a multiple knowledge base approach to AUV mission control [Ref 21], object-oriented design techniques are employed to allow programs to be highly modular and to be distributed on

EXPECTED TECHNOLOGY ADVANCEMENTS

****Increased Speed on Conventional Machines:*** Better LISP and C optimization is increasing the development and run speed of expert systems on all machines [Refs. 22, 23, 24].
****Supercomputers on a Chip:*** Very high speed integrated circuit (VHSIC) technology makes it possible to have micro-chips that contain about 35 million transistors which will make almost instantaneous processing of expert systems possible at workstations [Refs. 25, 26]
****Parallel Processing:*** New computers are being developed with over 64,000 processors acting in parallel. By working parts of problems simultaneously, machines will be able to find solutions faster[Ref. 27].
****Neural Networks:*** Research is proceeding in the effort to use metal-oxide semiconductor field-effect transistors to emulate the way simple biological networks work. Development in this area will revolutionize the "thinking machine" concept used by autonomous systems [Refs. 28, 29, 30, 31].

Figure 2.4 Expected Technology Advances

multiple workstations which communicate across interprocess communications links. Use of these special purpose workstations, as opposed to general purpose computers, provide the best environment for development of complex simulation and control software. As such, a "laboratory environment" has been produced at NPS that allows rapid prototyping of AUV planning, navigation, and control subsystems which ultimately reduces the overall time and expense of AUV subsystem development.

In short, AI workstations provide an excellent environment for the development of large-scale complex programs that model human intelligence and behavior. Special purpose processors allow for high-speed symbolic processing using LISP or PROLOG programming languages. High-speed, high resolution 3-D visualization systems allow the production of realistic dynamic models of systems under development. Expert system shells allow the development of extremely useful and "user-friendly" man-machine interfaces. Networking facilities, large memory capacity, and sophisticated memory management provide the necessary flexibility and speed for the development of intelligent, dynamic simulators and development laboratories [Ref. 8:p 17].

E. SUMMARY

This chapter presents background information on previous research that is relevant to this thesis. A brief synopsis of the historical development trends of both Autonomous Underwater Vehicles and Expert Systems is addressed and is followed by an overview of recent developments in both areas of research. This chapter is concluded with a brief discussion of AUV research conducted at NPS in the Computer Science area of study, and the outstanding utility of using distributed, single-user workstations in this research.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

This research is part of an inter-departmental project at the Naval Postgraduate School that is in the process of designing and building an experimental Autonomous Underwater Vehicle. The purpose of this thesis is to enhance existing real-time, computer graphics simulations of the proposed AUV by upgrading mission profiles, improving the mission planner/dynamic model interface, and designing and producing an AUV Mission Planning and Control Workstation. Additional research is being conducted to determine advanced uses of knowledge-based expert systems in the development of AUV mission planning and control algorithms.

B. NPS AUV CHARACTERISTICS

The original NPS Autonomous Underwater Vehicle design is patterned after the U.S. Navy's Swimmer Delivery Vehicle (SDV) that is used for the covert delivery and extraction of Special Warfare teams into and out of sensitive areas [Refs. 8,9]. Use of this model provided an initial vehicle dynamics database from which to start the NPS design efforts. Development, building, and testing of the NPS Model 1 AUV provided additional hydrodynamic data from which to refine the design, and currently, the NPS Model 2 AUV is under construction.

1. NPS Model 1 AUV

The NPS Model 1 AUV is a small 30 by 7 by 3.5 inch tethered model that was used to generate much of the initial hydrodynamic and control subsystem data used for the design of the NPS Model 2 AUV, which is currently under construction.

As reported by Boncal [Ref. 33:p. 102], this model was built to aid in the design of a model following autopilot that could be used in an Autonomous Underwater Vehicle. As an initial database, the SDV hull shape and hydrodynamic characteristics were well studied, documented, and displayed many of the attributes of a potential AUV.

The model was built as a dramatically scaled-down version of the SDV and a 19 state controller was designed for automatic control of maneuvers in the dive plane only. This controller design displayed fairly robust control and trajectory following characteristics over a five to one speed range and provided an excellent array of hydrodynamic test data for use in the more advanced controller designs planned for the NPS Model 2 AUV [Ref. 33:p. 103].

2. NPS Model 2 AUV

The basic shape of the NPS Model 2 AUV is also similar to the SDV. The model under construction will be a 350 lb (approximate), 84 by 16 by 10 inches, flattened cylinder hull shape with a rounded bow and tapered stern. The AUV will maneuver with bow planes, stern planes, twin rudders, and twin screws. In addition, the model is being equipped with two vertical and two horizontal tunnel thrusters for hovering and maneuverability at low speeds. Figure 3.1 is a simple line drawing of the proposed body and control surfaces arrangement.

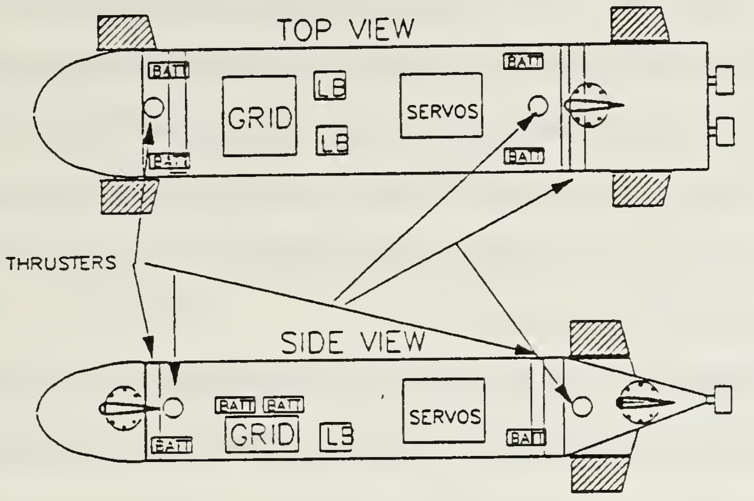


Figure 3.1 Line Drawing of NPS Model 2 AUV

Three generations of autopilot designs have been tested for possible use in the NPS Model 2 AUV: Proportional-Integral-Derivative (PID) Control, Sliding Mode Control (SMC), and Adaptive Variable Structure Control (AVSC). Each of these designs are much more robust systems than the controller used in the earlier model, with the latter two designs offering the most promise.

The PID controller is a simple, not very robust, non-adaptive (model specific) design scheme that uses system gain adjustments to provide the desired response [Ref. 34:p 260]. The SMC design is much more robust and is a significant improvement over the PID design. The SMC principle, in short, is based on a sliding plane and a switching logic that stabilizes, asymptotically, pairs of unstable structures in the control system. This principle results is a very fast and accurate system with a relatively wide bandwidth. However, this system is also non-adaptive [Ref. 35:p. 212]. The AVSC uses the same principles as the SMC but is more robust and not model specific, to a great extent. This system is based on using adaptive gains which guarantee convergence of the state vector to the sliding surface [Ref. 35:pp. 213-215].

To test these control systems in a more realistic environment, 3-D, dynamic, visual simulators of the proposed AUV have been developed.

C. NPS AUV SIMULATORS

Design and development of new systems, such as autonomous underwater vehicles, can be an expensive and time consuming process without a means of rapid prototyping and concurrent testing of algorithms, subsystems, and high-level mission planners [Ref. 9]. Researchers at NPS seek to solve this problem through the use of 3-D visual simulation. Previous simulator research at NPS by MacPherson [Ref. 8] has shown that graphics workstations provide a useful way to simulate a realistic external environment for conducting AUV operations and, more recently, Nordman [Ref. 9] has developed a simulator that generates a "laboratory environment" for testing several AUV planning, navigation, and control subsystems. This approach permits the prompt development and thorough testing of AI software for the NPS Model 2 AUV, the examination of different AUV hydrodynamic models, the testing of maneuvering subsystems in conjunction with different sensor configurations, as well as the rapid prototyping and development of AUV Mission Planning and Control Workstations for possible use on AUV launching platforms.

At present, three different simulators have been developed in the progression to provide more and more useful and versatile rapid prototyping and development tools for producing mission planning and control subsystems for the NPS Model 2 AUV.

1. NPS AUV-SIM1

This 3-D visual simulation system was developed before the decision was made as to the vehicle body shape of the NPS AUV, but was the first attempt at approximating expected AUV behavior in an open-ocean mission environment. As such, this system permits mission execution without a detailed implementation of AUV dynamics. The simulator represents a small manned vehicle with a control panel and a "through the periscope" display. The NPS AUV-Sim1 dynamics model can be operated manually or in the autopilot control mode and consists of a simple point-mass approximation governed by one acceleration equation, two rate equations, and one attitude equation. The vehicle's location, orientation, and motion characteristics are represented 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 by the onset of cavitation. The vehicle's pitch angle is determined by the AUV's speed and sternplane angle and it's acceleration is fixed at 1 knot/sec^2 while depth and azimuth rates depend on a combination of speed and control surface angle. [Ref. 8]

Currently, this simulator is used infrequently, but has been retained for use during future, more advanced open-ocean mission development for the NPS AUV. However, the simulator's mission control software and structure, developed using the KEE knowledge-based expert system [Ref. 8], provided an excellent framework from which to develop more advanced mission planning and control systems. Its control structure is a hierarchical system architecture that divides control among three areas: the mission level, the planning level, and the execution level. This structure has been used to develop the NPS AUV-SIM2 and 3 and will also be the blueprint for the actual onboard control system for the NPS Model 2 AUV. This hierarchical structure will be discussed in more detail in the next chapter.

2. NPS AUV-SIM2

The NPS AUV-SIM2 system developed by Nordman [Ref. 9] utilizes a 3-D visualization model of the proposed NPS AUV based on the SDV's dynamics and shape and on preliminary NPS hydrodynamic test data gained from experimentation with the NPS Model 1 AUV [Ref. 32]. The water environment for this simulator is modelled after the proposed initial test site for the NPS Model 2 AUV, which is a swimming pool measuring approximately 120 by 60 by 8 feet. Figures 3.2 and 3.3 are 3-D simulator views of the dynamic model and represents the initial design of the NPS AUV. The actual vehicle will differ somewhat from this simulator view due to a new rudder design and a slightly different body shape.

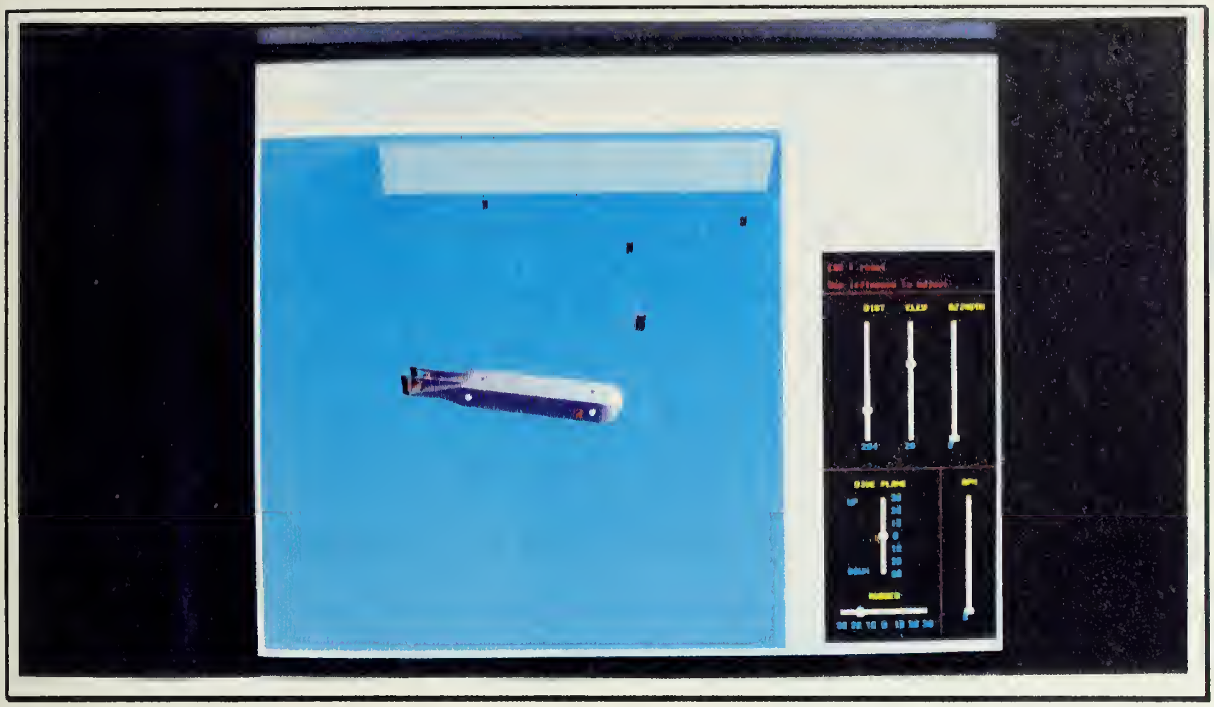


Figure 3.2 NPS AUV-SIM2

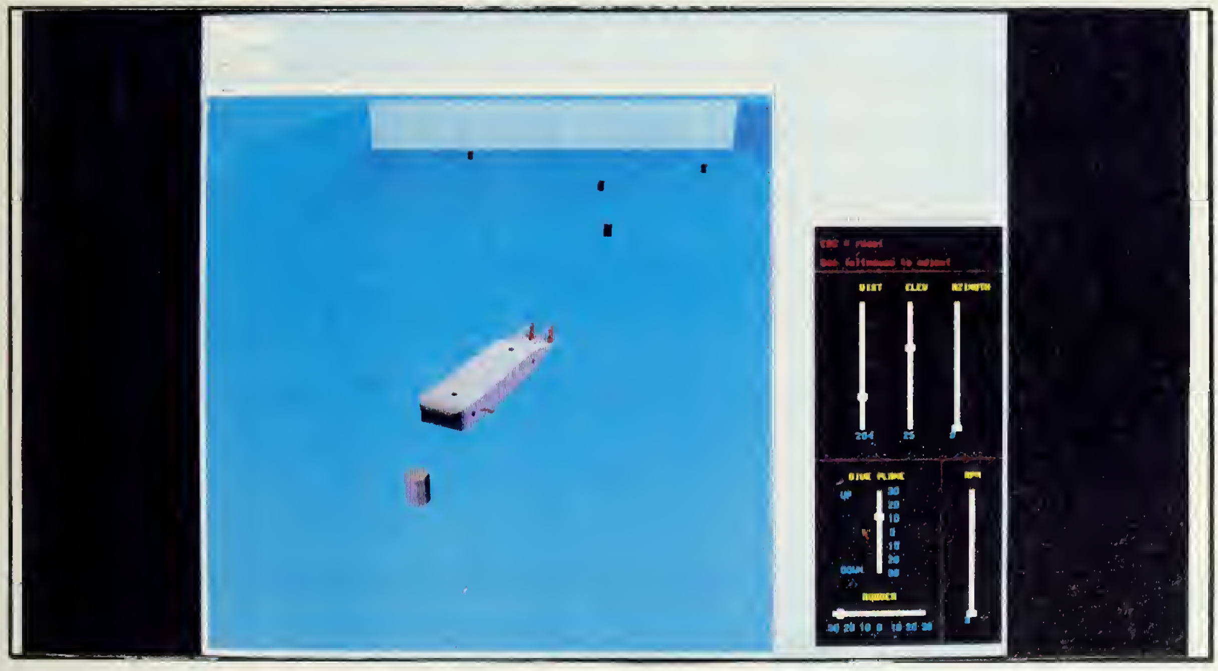


Figure 3.3 NPS AUV-SIM2

This system uses a simple depth or course-error calculation to set control surface positions for maneuvers. Manually selected or autopilot orders create control surface angles which in turn act on the AUV hydrodynamic model to generate hull pitch angles and resulting changes in depth and course. This system uses a first order, PID controller that produces abrupt and non-linear control surface behavior. However, as more advanced control structures are developed, the system's modular code structure allows easy installation between the autopilot and the hydrodynamic model for testing and analysis.

This simulator system retains the hierarchical mission planning and control structure that was developed for NPS AUV-SIM1. The mission planner and controller controls execution of missions by the dynamic model by transmitting orders and receiving positional data across interprocess communications links. In this format, the simulator provides an excellent "laboratory environment" for prototyping and development of advanced AUV planning, navigation, and control subsystems. The original version of this simulation system utilized a simplified mission selection, planning, and control structure designed with the KEE expert system shell. This thesis research further upgrades this simulation system by expanding use of the expert system shell to develop an AUV Mission Planning and Control Workstation as a prototype for use on an AUV deployment vehicle.

3. NPS AUV-SIM3

This simulator is identical to the NPS AUV-SIM2, with the exception that the AUV dynamics model has been upgraded with better

graphics drawing algorithms and a more advanced controller. The dynamic model used with this simulator system currently uses a Sliding Mode Control (SMC) system, but is also being used for experimentation with the Adaptive Variable Structure Control (AVSC) system.

One part of this thesis research was to enable this model to be driven in the autopilot (simulated autonomous) mode in the same manner as NPS AUV-SIM2. This system takes different inputs to operate its control surfaces, thus requiring an interface upgrade to couple the computer system that houses the 3-D dynamic model with the system that houses the mission planning and control system. The interprocess communications software used with NPS AUV-SIM2 was also used with this system with only minor modifications. This interface was also modularized to facilitate ease of alteration if the need arises during future research.

This simulator is primarily used for testing of new controller algorithms and hydrodynamic equations.

D. MISSION PLANNING AND CONTROL WORKSTATION

As reported by Bane and Ferguson [Ref. 34] and more recently, by Nordman [Ref. 9], the U.S. military has identified well over 70 military missions especially suited for unmanned submersibles, in particular, autonomous unmanned submersibles. Concurrent with development of these autonomous systems, an equal amount of thought and effort must be devoted to developing comprehensive, easy-to-use, and fully interactive mission planning and control workstations for use by operational personnel. These systems must facilitate easy as well as rapid response programming to meet the particular mission scenarios at hand.

This area of AUV development has not been fully explored during previous research at NPS and is a major portion of this thesis research effort. The goal of this effort was the development of a robust and user-friendly AUV Mission Planning and Control Workstation that could be used as a prototype for possible use aboard AUV deployment platforms.

Development of this workstation starts with the mission planning and control structure developed by MacPherson and Nordman [Refs. 8,9] and extends the use of the KEE expert system shell to produce a system that is completely mouse and message driven. The workstation contains mouse actuated method units for AUV deployment/recovery actions, communication channel selection, type mission selection, and goal parameter selection. After mission and goal parameters are selected and the current location of the AUV (xy-coordinates and depth-under-the-auv) is obtained, the data is sent to a path-planner that uses a best-first search algorithm to plan the mission path. A Mission-Plan display panel shows mission parameters after the path planning has been completed, to include start location, goal location, and subgoal locations along the path. During mission execution, additional display units show orders sent to the AUV (ordered course, depth, and speed), and data received from the AUV, to include, positional data (xy-coordinates and depth-under-the-auv), actual heading, actual depth, and rpm. This latter information is transmitted over interprocess communication channels as part of the simulation system and represents data that might be transmitted via fiber-optic cabling between a deployed AUV and a deploying platform. The NPS Model 2 AUV will be outfitted to operate with or without a fiber-optic cable for data flow back to the deployment site, but will receive movement orders from a pre-programmed on-board computer. Additionally, information messages are printed to a typescript window to guide the user each step of the way during the mission planning and execution process.

A more detailed description of this workstation is included in the next chapter.

E. MISSIONS UPGRADE

The missions programmed into the NPS AUV simulators are described in detail by Nordman [Ref. 9]. These missions are divided into four main categories: payload/transponder delivery, charting, reconnaissance, and surveillance. A fifth category, "test.pool", was added by Nordman to NPS AUV-SIM2 and 3 to facilitate testing the dynamic model of the NPS Model 2 AUV in a representative water environment. The code for this mission and supporting functions has been modified to facilitate data flow for operation and testing of the Mission Planning and Control Workstation. The other missions are designed for testing AUV response in the open-ocean environment (NPS AUV-SIM1) and have not been upgraded to work in the test pool environment. This upgrade has been reserved for future research.

F. MISSION PLANNER/DYNAMIC MODEL INTERFACE

During the development of NPS AUV-SIM3, it was desired to enable the enhanced dynamic model to be driven in the autopilot mode in the same manner as NPS AUV-SIM2. As this new model uses a different controller from that used with the earlier simulator, and requires different inputs to control the model's control surfaces, it was necessary to modify/upgrade the interface between the computer graphics system and the mission planning and control system to simulate autopilot control. This process proved to be extremely cumbersome and time consuming and involved modification of a sizable portion of code in the AUV dynamic model's primary graphics file with a few additional modifications to supporting files. To make it easier to interface future 3-D graphics models with the mission planning and control system, the code within the graphics file that is required for this interface has been converted to a modular format. This will allow development of any new interface requirements to be completed "outside" the main body of code, with a simple and rapid swap of modules for testing of the new system.

G. SUMMARY

This chapter provides a detailed discussion of the problems considered for this study and the approach to providing solutions. A very general description of the evolution of the NPS AUV is included as well as a description of the simulator systems that have been developed to facilitate testing of certain planning and control sub-systems in a realistic "laboratory environment." Additionally, the characteristics and utility of the Mission Planning and Control Workstation are presented.

IV. AUV SIMULATOR DESCRIPTION

A. INTRODUCTION

This chapter describes, in detail, the NPS AUV-SIM2 simulation software with the addition of the Mission Planning and Control Workstation. The NPS AUV-SIM1 was described in detail by MacPherson [Ref. 8] and Nordman [Ref. 9] and its characteristics were summarized in the previous chapter. Because of its limited use at this point in the NPS AUV development, it will not be discussed in any detail in this chapter.

The description starts with an overview of the hierarchical AUV software architecture and how each level of the structure carries out its assigned tasks. This structure was described in detail by Nordman [Ref. 9], but portions of that description must be repeated here for clarity and continuity.

The last section of this chapter is a user's manual which repeats portions of the Nordman [Ref. 9] manual, also for clarity and continuity.

B. SIMULATION FACILITIES

As previously discussed, all three simulator systems run on interconnected workstations that communicate across interprocess communications lines. The systems used are LISP machines for mission planning and control and IRIS graphics machines for 3-D visualization of dynamic models and mission execution.

The Symbolics 3675 and Texas Instruments (TI) Explorer II LISP machines were available for mission and planning level simulator control and the IRIS 4D/70GT and the Silicon Graphics IRIS-2400T graphics workstations were available for the simulator execution level and 3-D display. The older IRIS-2400T graphics workstation is used for the NPS AUV-SIM1 simulator system and can be controlled in the autopilot mode from either the Symbolics or the TI Explorer LISP machines. Operation of this combination is described in detail by Nordman [Ref. 9]. The Symbolics LISP machine and IRIS 4D/70GT graphics workstation combination was chosen as the primary configuration for NPS AUV-SIM2 and 3, and for development of the Mission Planning and Control Workstation.

1. Symbolics LISP Machine

The LISP machine is aptly described by Nordman [Ref. 9]. "This machine is an advanced single-user workstation that uses the KEE software development shell to support development of large-scale and complex artificial intelligence programs. The programming environment includes very high speed proprietary processors, a large memory, sophisticated caching and memory-management system, high-resolution black-and-white graphics, support for color image processing, and networking facilities. The KEE shell gives the operator a productive and intuitive programming environment for developing large and complex applications."

The Symbolics 3675 was chosen over the TI for development of the NPS AUV simulation systems because it incorporates a Pixar image processing computer which offers significant advantages for future research and also possesses a color terminal suitable for graphical representation of mission execution. The TI LISP machine offers neither of these features [Ref. 9].

2. Silicon Graphics IRIS 4D/70GT

This system architecture uses multiple RISC-based CPUs with a high-speed 64-bit data bus and a 96-bitplane raster subsystem. In addition to a very fast hardware Geometry Engine, the Unix-based software supports high speed image generation and updating for object-oriented programming. The system readily supports a higher update rate for a real-time AUV simulation while simultaneously incorporating graphics lighting and shading models [Ref. 9].

C. CONTROL SYSTEM ARCHITECTURE AND LANGUAGES

This thesis has preserved the hierarchical system architecture implemented by MacPherson [Ref. 8] for development of the Mission Planning and Control Workstation. This architecture also provides the basic framework for the software structure of the NPS Model 2 AUV.

The software system divides control among three areas: the mission level, the planning level, and the execution level, as can be seen in Figure 4.1. The top level of this architecture is the mission level--a knowledge base implemented using the KEE software development shell. This knowledge base has been expanded to include a Mission Planning and Control Workstation. This workstation is composed of two parts; a Mission Planning and Control Panel (Figure 4.2) and an external color display monitor. The panel allows selection of certain parameters at the mission level and provides display units for monitoring the results of software operations during the planning level and real-time feedback data from the 3-D dynamic model during the execution level. The external display monitor provides a 2-D representation of the AUV's track during mission execution.

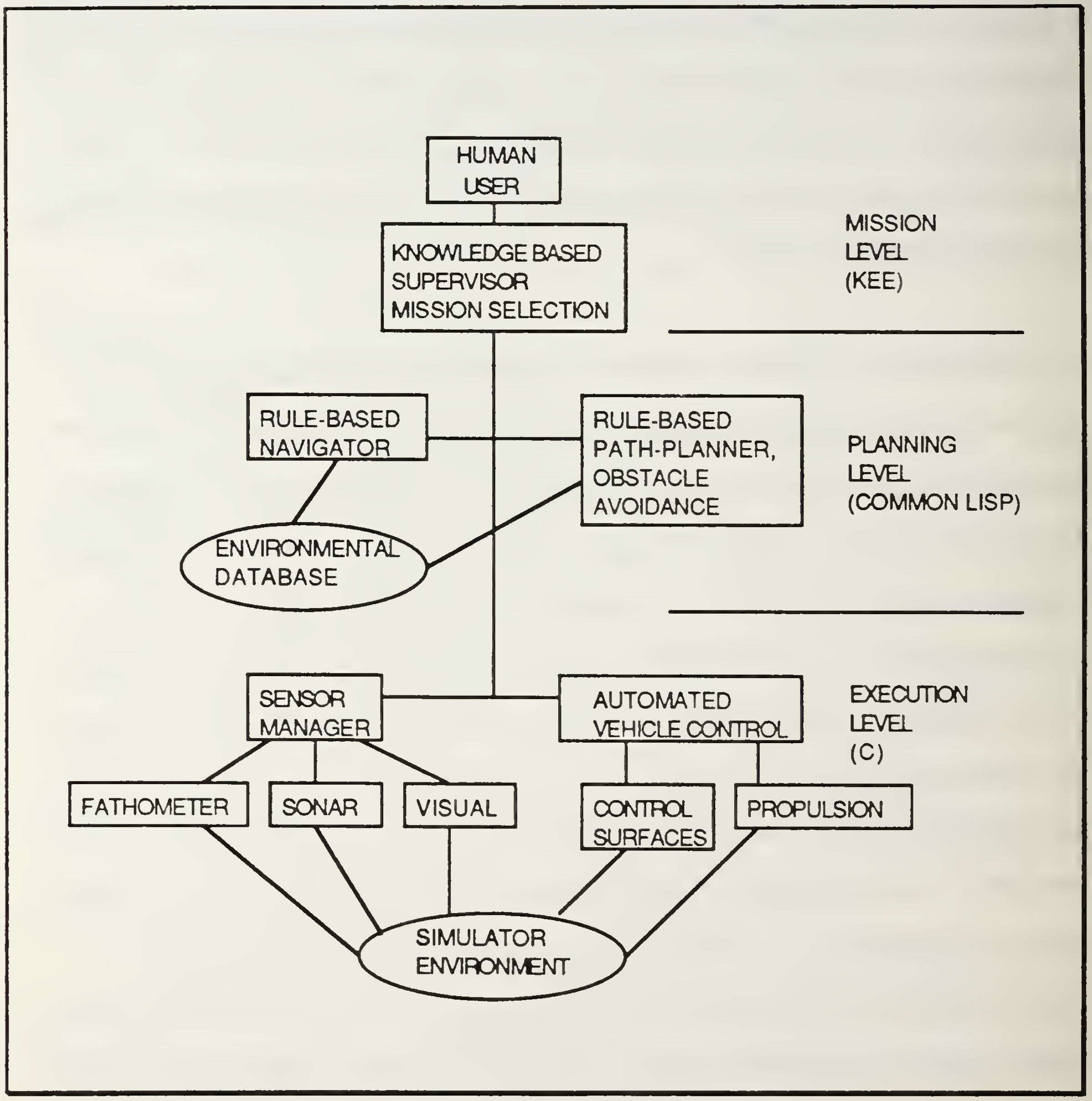
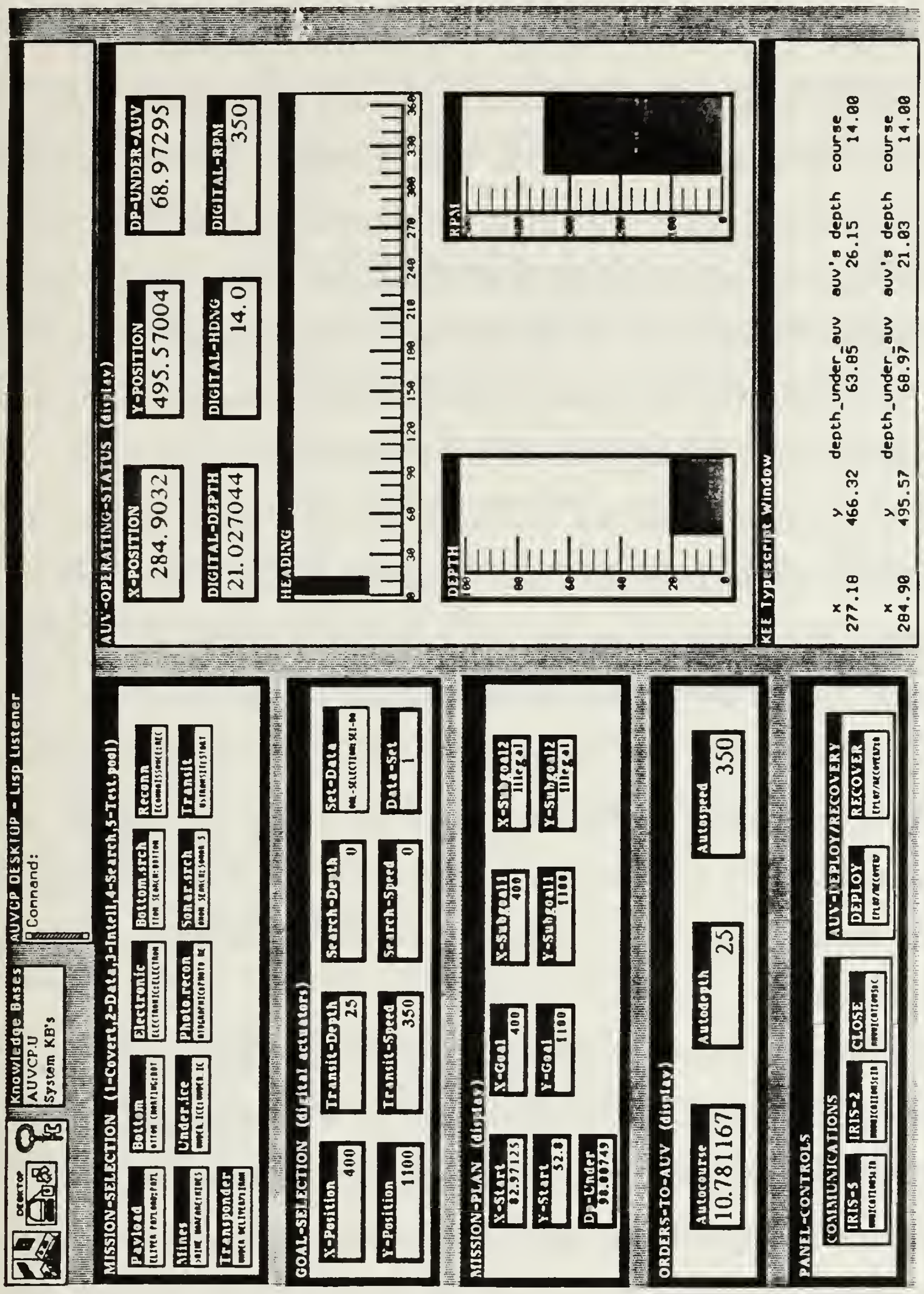


Figure 4.1 Hierarchical System Architecture



GOAL-SELECTION (digital actuators)

X-Position 400	Transit-Depth 25	Search-Depth 0	Set-Data
Y-Position 1100	Transit-Speed 350	Search-Speed 0	Data-Set 1

MISSION-PLAN (display)

X-Start 82.97125	X-Goal 400	X-Subgoal1 400	X-Subgoal2 Illegal
Y-Start 52.8	Y-Goal 1100	Y-Subgoal1 1100	Y-Subgoal2 Illegal
Dp-Under 98.00749			

ORDERS-TO-AUV (display)

Autocourse 10.781167	Autodepth 25	Autospeed 350

AUV-OPERATING-STATUS (display)

X-POSITION 284.9032	Y-POSITION 495.57004	DP-UNDER-AUV 68.97295
DIGITAL-DEPTH 21.027044	DIGITAL-HDNG 14.0	DIGITAL-RPM 350

KEE Typescript Window

```
   x         y      depth_under_auv   auv's depth   course
277.18    466.32        63.85            26.15        14.00

   x         y      depth_under_auv   auv's depth   course
284.90    495.57        68.97            21.03        14.00
```

Figure 4.2 Mission Planning and Control Panel

Operations begin with the operator interacting directly with the knowledge base by selecting "deploy" on the AUV-Deploy/Recover sub-panel. This action initializes all display units and prompts the operator to set up the 3-D dynamic model in the "autopilot" mode, select the appropriate communication channel to the graphics workstation, select the desired mission, and select desired goal parameters. Once these steps have been completed, the operator is prompted to access the current location and depth of the AUV. This information along with the mission selection data is then passed to the planning level for the mission planning process.

After the required mission selection information is passed to the planning level, the path-planner and navigator plan the best path to the desired goal. The path-planner and navigator are Common LISP software modules that consult the environmental database for the location of known obstacles, and then use a best-first search algorithm to plan the desired path. Upon completion of the path planning process, this path, which consists of the starting location, the goal location, and a series of subgoals, is reported to the mission level and is displayed on the Mission Planning and Control Panel.

Once supplied with the mission parameters and a path to the mission's goal, the planning level prompts the operator to actuate the third level--the execution level. Maneuvering parameters in the form of autocourse, autodepth, and autospeed, are forwarded to the graphics workstation via the communications network and are interpreted by the execution-level autopilot as control surface commands that put the simulator's AUV on the path's course, speed, and depth [Ref. 9]. The execution level software includes provisions for sensor modules that can provide simulated electronic, acoustic, and visual environmental inputs to the AUV[3] . These inputs, as well as AUV position data, is passed back up the hierarchy to the navigator where the data is analyzed, displayed on the control panel, and, if necessary, acted on.

AUV position data is also passed by the navigator to an external color monitor that displays the AUV track as the mission is being executed. This display consists of a 2-D scale representation of the test pool environment, where multi-colored icons represent the AUV's start location, goal location, and track. The display monitor track, which is retained for analysis until "cleared", also provides an additional means of analyzing the path-following accuracy of the AUV.

Once the mission is complete, the operator is prompted to close the communications channel, to recover the AUV (re-initializes the control panel display units), and to save and delete the knowledge base, if no more mission runs are desired.

1. The Mission Level

The KEE software development shell was used to develop the mission level. KEE's representational features, reasoning and analysis systems, and access to knowledge with KEE graphics tools offer many advantages in developing applications, not the least of which, is the ability

[3] The NPS AUV-SIM1 simulator system utilizes these simulated sensor inputs for the open-ocean missions. However, the NPS AUV-SIM2 and 3 systems have been programmed for the test pool environment only and do not, as yet, use these inputs.

to build functional application prototypes, complete with user interfaces, in a short period of time [Ref. 37].

The structure of this knowledge base is made up of image panels, frames (units), methods, and active images. A knowledge base can contain behavioral, as well as, descriptive knowledge. Behavioral knowledge is represented in the KEE system by methods. Each method is a small LISP program, and whenever the method is called, the program is run. The method carries out whatever task it is instructed to do and can run from the very simple to very complex algorithms.

In this simulation system, the mission planning and control panel is made up of active images, which are either mouse actuated images that call methods, or display units that display analog, digital, or literal data. Image panels are used to group or organize related active images, and, in some cases, contain sub-panels that also contain active images.

As described in the previous section, missions are normally selected by mouse activating the appropriate active image. The structure of this knowledge base also graphically links related portions of the knowledge base on a tree diagram that allows inheritance of information down the hierarchy from parent to child nodes. This facility also allows activation of the missions by selecting ,with the mouse, the desired mission at the leaf nodes of the tree. After a selection is made in this manner, the operator is prompted for additional information that is used to plan and execute the mission. This, however, is not the recommended way to operate the simulation system. Figure 4.3 shows the graph of the knowledge base developed for this simulation system.

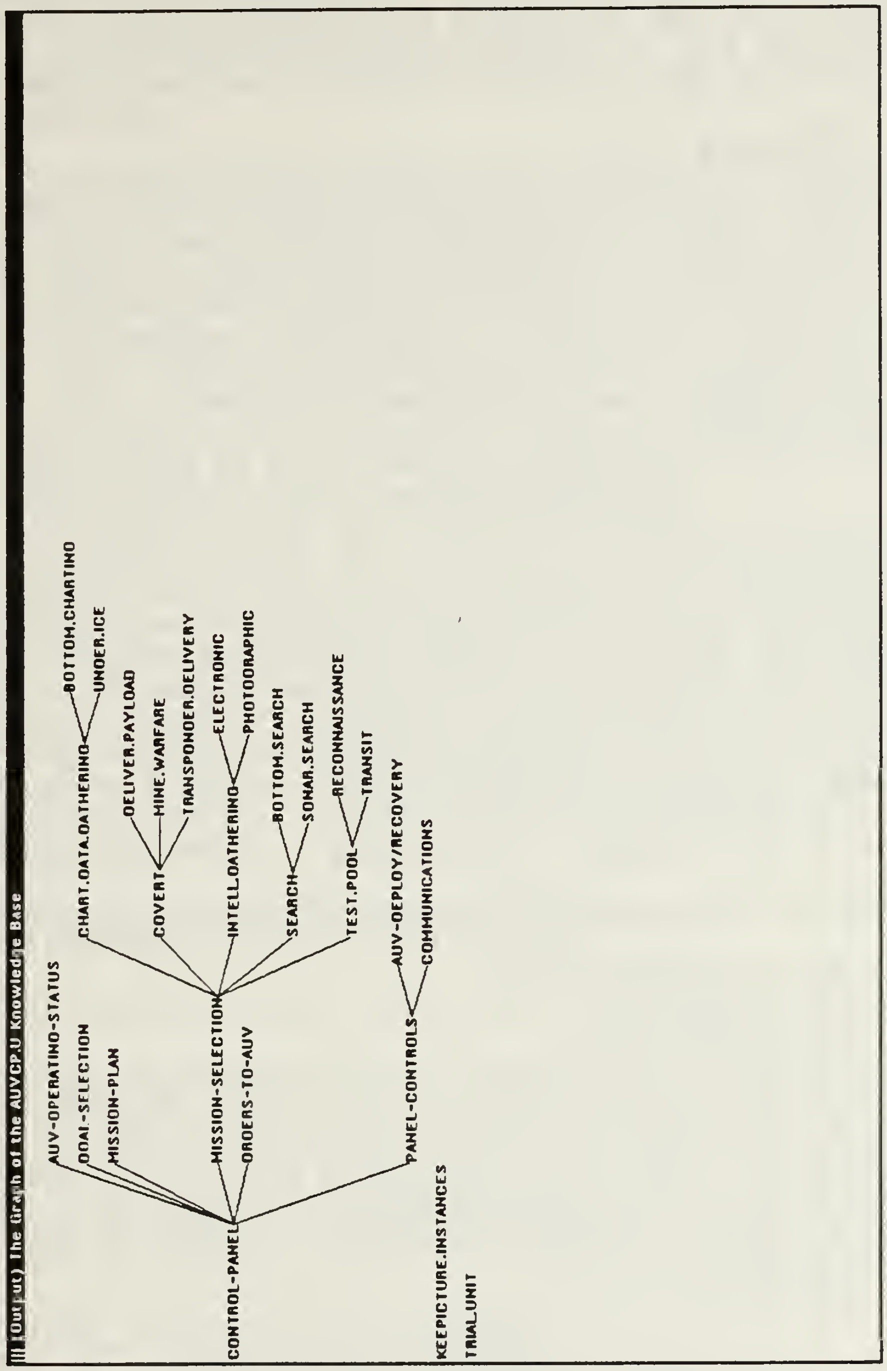


Figure 4.3 Graph of The AUV Knowledge Base

Currently, only the test.pool "transit" mission has been fully programmed for use with the Mission Planning and Control Workstation and is used to test the dynamic model's propulsion and control subsystems.

2. The Planning Level

The planning level is also written in Common LISP and runs on the Symbolics LISP machine for this simulator. As described earlier, this level consists of a number of software modules that compose the mission navigator, path planner, and environmental database. After the path planner receives mission selection data and plans an appropriate mission path, the navigator receives orders from the mission level and provides corresponding guidance commands to the execution level. Using frequent data exchanges via the communications interface, the navigator provides the execution level with information on the next subgoal, the autopilot course, speed, and depth, and the commands required to execute the current phase of the mission. The planning level receives sensor and positional data from the execution level and modifies mission commands as necessary for the next data exchange.

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 and executes either manual or autopilot commands to update the vehicle and environmental displays.

In the manual mode, in addition to being able to "fly" the AUV model by varying the control surface positions, the operator can change the viewer's perspective of the environment by manipulating mouse controlled slider bars on the control panel to vary the viewing distance, elevation, and azimuth. This feature is available on both the NPS AUV-SIM2 and 3 simulation systems.

The control surfaces of the AUV model in the NPS AUV-SIM2 simulation system are varied by assigning desired angles. The NPS AUV-SIM3 simulation system differs in that the AUV model takes ordered headings (+180 degrees to -180 degrees) and ordered depths (150 units up to 150 units down) and the control surfaces are programmed to move appropriately to achieve the ordered heading/depth.

In the autopilot mode, the execution level interprets planning level commands and positions the AUV control surfaces to achieve the ordered parameters. At each update of the 3-D visual display, the execution level passes sensor and position data up the hierarchy to the planning level where it is displayed on the control panel and acted on, if appropriate.

D. COMMUNICATIONS SOFTWARE

During execution-level operations, interprocess communications support must be available for data transfer between the execution-level code on the IRIS graphics machine and the planning-level on the LISP machine. Communications modules in the code of each workstation link the systems via an Ethernet cable; each module passes the same data types and structures in slightly different formats.

Figure 4.4 shows the communications flow of information between the two workstations. The diagram shows both workstations having individual send and receive ports. Actually, two ports are used on the LISP machine

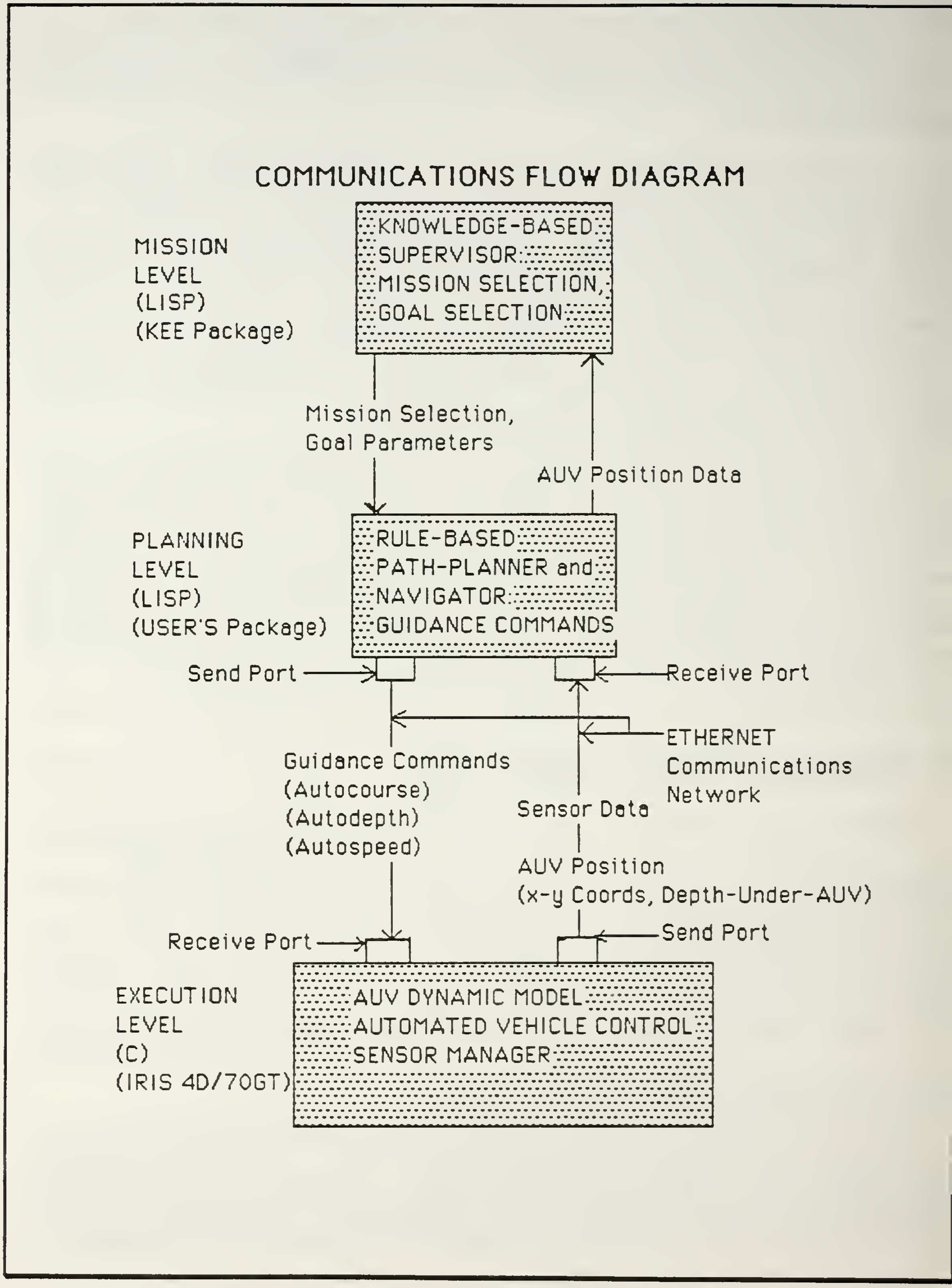

COMMUNICATIONS FLOW DIAGRAM
MISSION LEVEL (LISP) (KEE Package)
KNOWLEDGE-BASED SUPERVISOR MISSION SELECTION, GOAL SELECTION
Mission Selection, Goal Parameters
AUV Position Data
PLANNING LEVEL (LISP) (USER'S Package)
RULE-BASED PATH-PLANNER and NAVIGATOR GUIDANCE COMMANDS
Send Port
Receive Port
ETHERNET Communications Network
Guidance Commands (Autocourse) (Autodepth) (Autospeed)
Sensor Data
AUV Position (x-y Coords, Depth-Under-AUV)
Receive Port
Send Port
EXECUTION LEVEL (C) (IRIS 4D/70GT)
AUV DYNAMIC MODEL AUTOMATED VEHICLE CONTROL SENSOR MANAGER

Figure 4.4 Communications Flow Diagram

but all data is transferred through a single port (dual socket) on the IRIS graphics workstation. The drawing arrangement has been simplified for clarity.

The Mission Planning and Control Panel allows the operator to select one of two communications paths: IRIS-5 for NPS AUV-SIM2 and 3, or IRIS-3 for NPS AUV-SIM1. The operator selects the machine on which the communications will be run, which determines what portions of the communications modules will be used to support the simulation. The data exchange between the LISP machine and the IRIS graphics workstation allows the planning level to pass guidance commands to the execution-level which controls the actions of the AUV dynamic model during execution of the mission. The guidance commands are autocourse, autodepth, and autospeed. The execution-level, in turn, passes sensor and AUV position data over the communications network to the planning level for information display and actions by the navigator, if required by the mission profile. Data transfer occurs approximately every three seconds during the execution-level process.

The communications code is adapted from MacPherson [Ref. 8] for NPS AUV-SIM1 and from Nordman [Ref. 9] for NPS AUV-SIM2 and 3. This thesis research found no need and made no attempt to modify these communications systems.

E. USER'S MANUAL

The NPS AUV-SIM2 and 3 simulators are completely mouse and message driven. Messages printed in the typescript window of the Mission

Planning and Control Panel prompt the operator for input that can be accomplished with mouse actuation of certain active images on the panel. This manual assumes some basic familiarity with the Symbolics LISP machine and the IRIS graphics workstation. Some very basic experience is required with the KEE expert system shell and the UNIX operating system for startup and shutdown of the simulator, but not for its operation.

NPS AUV-SIM1 is discussed in detail by Nordman [Ref. 9] and will not be covered in this manual. This manual does repeat portions of the Nordman [Ref. 9] user's manual for continuity.

1. Graphics Workstation Operation--IRIS 4D/70GT

a. NPS AUV-SIM2

The NPS AUV dynamic model may be operated in either the manual or the autopilot modes. This simulation is normally run on the IRIS 4D/70GT (IRIS-5) because of its close proximity to the LISP workstation, which allows easy viewing of both workstations during the autopilot mode of operation. However, the IRIS machines are networked in a manner that allows the simulation to be run on either IRIS-1, IRIS-4, or IRIS-5.

To start the simulation, "log on" to both the IRIS workstation and the side terminal of the IRIS and transfer to the directory */usr/work/nordman/symbolics/auvsim*. Start the program in the manual mode by entering the command *auv* on the side terminal followed by a carriage return.

The simulated AUV starts *on the surface* at a speed of *25 rpm* on course *east*. All manual control in this simulator uses the mouse to manipulate sliding markers on the control panel at the right side of the main terminal display. To alter the viewer's perspective or to change AUV parameters, press and hold the left mouse button, and drag the marker to the desired new value for that parameter. (Changes to the viewer's perspective should be executed slowly or the user may lose his own perspective in the display.) Care should also be taken when manipulating the markers that change the control surface angles. The sliding markers for the rudder and the dive planes[4] are tied together and can be operated simultaneously to *fly* the AUV around the test pool environment. To operate the rudder independently of the planes, the mouse cursor must be placed at the *0 degree* level of the dive plane control and moved left or right for independent rudder control. For simultaneous operation of control surfaces, simply drag the mouse cursor to the desired dive plane angle, and while keeping the cursor at the desired level, move the cursor left or right of the dive plane control bar for simultaneous rudder operation.

At very low speeds, the AUV may slowly roll from side to side. This is caused by an instability in the mathematical model at low speeds and raising speed will restore proper control surface effects on the hydrodynamic drag model and should damp out this motion.

---

[4] The dive planes consist of both bow and stern planes. When the dive planes angle is changed using the mouse-controlled sliding marker, the bow and stern planes move to the requested angle but in opposite directions. This is a simplified representation of how a submarine would operate with bow and stern planes.

The autopilot mode is started by pressing the A-key on the main keyboard of the IRIS machine. The AUV should be driven to the desired starting location prior to activating the autopilot mode. (The AUV can be returned to the original starting position of the manual mode by pressing the ESC-key.) Additionally, once the autopilot mode has been activated, the display must be secured and restarted to return to the manual mode of operation. After activating the autopilot mode, the side terminal will indicate that the IRIS server is waiting to connect to sym1 (the Symbolics LISP machine) and the following message will prompt the operator to start the KEE portion of the simulator to connect the LISP client to the IRIS server:

```
Start the autopilot program on the lisp machine.
Then hit any IRIS key to send initial AUV position.
Server waiting to connect to SYM1
```

The autopilot execution can be interrupted by pressing the Q-key; the autopilot cannot be restarted at this point, but manual control of the AUV is available. Prior to restarting the IRIS 4D/70GT simulator autopilot, ensure the previous communication socket connection has been completely broken. To do this, list the current processes by entering the Unix command *ps* on the side terminal keyboard. A list of active processes and process numbers will appear. Stop any send/receive communications daemons with the *kill <process number>* command. (Inexperienced operators may require assistance for this step.)

b. NPS AUV-SIM3

This simulator is basically the same as NPS AUV-SIM2, with a few minor exceptions. To start the simulation, "log on" to both the IRIS workstation and the side terminal and transfer to the directory */usr/work/rogers/mythesis/mysymbolics/auvauto*. Start the program in the manual mode by entering the command *autoauvs* on the side terminal followed by a carriage return. Also, type the command *gclear* on the side terminal to initialize the mouse.

The simulated AUV starts *on the surface* at a speed of *250 rpm* on course *north*. Manual control in this simulator uses the mouse to manipulate sliding markers on its control panel as with the previously discussed simulator. However, the control surfaces for this simulator do not respond to ordered angles. AUV depth is changed by selecting an ordered depth (150 units up to 150 units down), and AUV heading is changed by selecting an ordered heading (-180 degrees to +180 degrees). The depth and heading sliding markers operate independently.

All other operations of this simulator are the same as those for NPS AUV-SIM2.

2. LISP Machine Operations--Symbolics 3675

a. NPS AUV-SIM2

On the Symbolics LISP machine, ensure that the KEE expert system shell software is loaded. (It is accessed by entering SELECT-K.) If the shell is not loaded, a cold boot of the machine will be required; an inexperienced user should refer to posted instructions near the machine or get staff assistance at this point. Once KEE is available, "log on" in the LISP Listener window and then press the SELECT-K combination on the keyboard to move to the KEE desktop. Ensure that the Symbolics external monitor is ready by depressing the "on" button, depressing the "degaussing" button and holding for at least 2 seconds, and turning the brightness knob fully clockwise. Once the KEE desktop is available, there are two methods of loading the knowledge base:

(1) Use the mouse and point the cursor at the latchkey icon located at the upper left corner of the screen and depress the left mouse button. A pop-up menu will appear offering *KEE Commands*; select the *Load KB* command by pointing at the command with the mouse cursor and depressing the left mouse button. A KEE "typescript" window will appear requesting the name of the knowledge base to be loaded; enter *sym4:>rogers>controlpanel-file>auvcp.u* followed by a carriage return. KEE will load the AUV Mission Planning and Control Panel knowledge base along with the LISP code files containing the planning level functions. (This process will take approximately 2 minutes.) Once the files have been loaded, a black icon will appear on the screen. (This is a prompt to enter an input by depressing one of the mouse buttons.) Depress the left mouse button and the image panels of the Mission Planning and Control Panel will appear one at a time. The external monitor will also expose a display window with a 2-D scaled grid pattern of the test pool environment. If the "typescript" and/or the "LISP listener" windows are not in the correct position and partially obscure the image panels, point the mouse cursor at the top bar of these windows and depress the right mouse button. A menu will appear that will allow the operator to select *move* and *reshape*. Select these options one at a time and move and reshape the windows as neccessary to clear the image panels. The Mission Planning and Control Workstation is now ready for operation.

(2) This method is somewhat slower than the first method but loads a preset desktop arrangement with the knowledge base. Use the mouse and point the cursor at the desktop icon located at the upper left corner of the screen and depress the left mouse button. A pop-up menu will pointing at the command with the mouse cursor and depressing the left mouse button. A KEE "typescript" window will appear requesting the name of the desktop to be loaded; enter *sym4:>rogers>controlpanel-file>auvcp.desktop* followed by a carriage return. KEE will load the AUV Mission Planning and Control Panel knowledge base along with the LISP code files containing the planning level functions. If an old desktop was on the screen upon logging on, it will need to be cleared before the auvcp desktop can be loaded. The "typescript" window will request the operator select one of the following options: *Respecify, Flush Old Desktop, Rename Desktop*, or *Debug*; enter *Flush Old Desktop* followed by a carriage return. The screen will then clear, after which, the pre-set desktop and the image panels for the control panel will appear. The Mission Planning and Control Panel is now ready for operation.

To start the simulation, select the *deploy* command by pointing the mouse cursor at the deploy method actuator in the Panel-Controls section of the control panel and depressing the left mouse button. This action will initialize all display units on the panel and cause a series of prompts to be printed in the KEE typescript window.

If not already accomplished, ensure the AUV portion of the simulator on the IRIS 4D/70GT has been placed in its autopilot mode and is waiting to connect to the Symbolics LISP machine *(Sym1)*.

Prior to selecting a mission, a communications channel must be opened to the IRIS workstation. Select *IRIS-5* by pointing the mouse cursor at the IRIS-5 method actuator on the Panel-Controls section of the control panel and depressing the left mouse button. This will start the TCP/IP software on the LISP machine and locate the correct IRIS port(s) for data exchange. Currently, IRIS-5 contains all the files for the graphics portion of this simulation system. (IRIS-3 contains the files for the graphics portion of NPS AUV-SIM1.) The following message should be appear in the KEE typescript window: *A conversation with the IRIS machine has been established.*

Select a mission by pointing the mouse cursor at the desired mission method actuator in the Mission-Selection section of the control panel. (Currently, only the *Transit* mission is fully programmed to work with this simulation system.) A series of messages will now appear in the KEE typescript window to prompt the operator to select appropriate goal parameters for the mission selected. Select goal parameters by pointing the mouse cursor at each of the required method actuators in the Goal-Selection section of the control panel, one at a time, and depressing the left mouse button. A calculator will appear on the screen to facilitate inputing the desired values. Select all desired numbers for a particular parameter value and then select *enter* with the mouse when the appropriate value has been entered in the calculator window. This action will cause the parameter value to appear in the appropriate parameter window on the control panel. This process must be repeated for each goal parameter required for the selected mission. Once the values for all parameters have been entered, select *Set-Data* by pointing to the method actuator with the mouse and depressing the left mouse button. This causes the goal data to be transmitted to the planning level of the simulator.

A prompt will now ask the operator to access the AUV initial position data from the IRIS workstation. This is accomplished by depressing any key on the IRIS keyboard. When this step is accomplished, the following message should appear on the screen of the IRIS side terminal:

> *Initial position written to lispmachine-hit any key when the lispmachine completes its search.*

Upon completion of this step, the *x-position*, *y-position*, and *depth-under-the-auv* should appear in the KEE typescript window of the Symbolics machine. This completes the data needed by the planning level to compute the mission path and this process starts without further action by the operator. When the mission path has been computed, the starting location, goal location, and subgoal locations will appear in the Mission-Plan section of the control panel and in the KEE typescript window, preceded by the following message: *autopilot course on the first leg is:* . Start autonomous simulator execution by pressing any key on the IRIS keyboard.

If necessary, the AUV simulation can be stopped by entering CONTROL-ABORT at the LISP machine or by pressing the Q-key on the IRIS keyboard. Once the simulator has completed its mission or after abortion of the mission, the Symbolics-IRIS communication socket connection must be closed by selecting *Close* on the Panel-Controls section of the control panel with the mouse and depressing the left mouse button.

The control panel should be reset at this point by selecting *Recover* on the Panel-Controls section of the control panel. This action re-initializes the control panel display units and the Symbolics external monitor, and a series of prompts will appear in the KEE typescript window to direct the operator in securing the knowledge base (if no more runs are desired). If the knowledge base has not been secured, the simulation may be restarted by securing and restarting the IRIS portion of the simulator, and repeating the steps listed above at the point of selecting a communications channel.

b. NPS AUV-SIM3

From the Symbolics workstation, the steps for activation and operation of this simulation system are exactly the same as those listed above for NPS AUV-SIM2.

I. SUMMARY

This section describes the NPS AUV-SIM2 and NPS AUV-SIM3 simulation systems, which includes the Mission Planning and Control Workstation. The description starts with an overview of simulation facilities followed by a detailed discussion of the control system architecture and the high-level programming languages used in the various levels of its hierarchy. A subsequent section gives a brief description of how the interprocess communications system is used for information exchange and control of the execution-level of the simulations. The final section of this chapter contains the User's Manual to assist in operating the various workstations to run these simulations.

V. EXPERIMENTAL RESULTS

A. INTRODUCTION

This chapter evaluates the utility of the Mission Planning and Control Workstation and presents experimental results relating to upgrades to the mission planner/dynamic model interface. All tests were conducted using the NPS AUV-SIM2 and 3 simulator systems.

B. AUV SIMULATION FACILITIES

1. The Original Mission Planner

The original mission planning module at the top level (mission level) of the software structure of the NPS AUV simulation systems was efficient but lacked the robustness and display features typical of operator's consoles for equipment as complex as autonomous systems.

After the original knowledge base was loaded, the AUV Simulator Mission Selection Tree was drawn in a window labelled *The Graph of the AUV Knowledge Base* . This graph was in a tree format similar to Figure 4.3, but only consisted of the mission-selection node at the top level of the tree and the generic missions located at the leaf nodes of the tree's branches. A mission was selected by using the mouse to point to the appropriate leaf node and pressing the left mouse button. This action would cause a pop-up menu of *Unit Commands* to appear. Selecting the *Send Message* option from this menu, with the mouse, would result in another pop-up menu of *Message Types* to appear. From this menu, the appropriate message that would start the selected mission would be selected and a series of messages, printed in the KEE typescript window, would prompt the operator to enter the required mission parameters from the keyboard.

When the typescript window announced that it had connected with the appropriate IRIS workstation, the current location of the AUV was accessed by the operator, and the path planning process was initiated when this data, along with the goal parameters, was passed to the rule-based path planner at the planning level. The remainder of the process is the same as that of NPS AUV-SIM2 and 3 (described in the previous chapter).

The procedure just described, though efficient, offered no feedback information to the operator and required more knowledge about the inner-workings of the KEE expert system shell than was felt necessary.

2. Mission Planning and Control Workstation

The concept of developing the Mission Planning and Control Workstation was conceived for two purposes: (1) to more fully investigate the facilities and advantages of the KEE software development tool, and (2) to create an informative, versatile, and easy-to-operate workstation as a prototype for use aboard AUV deployment platforms. To meet the second goal, it was desired that the workstation include clear and concise facilities for selecting missions, goal parameters, and communications channels, and include information display units to show the results of mission planning and AUV operating status (during mission execution). AUV operating status could possibly include pre-deployment readiness information, real-time feedback information from an AUV equipped with an electronic data link or fiber-optic cable, or expected track information based on the selected mission profile.

The Mission Planning and Control Console meets all of the aforementioned desires/requirements. The control panel includes sub-panels for mission selection, goal selection, communications channel selection, and selection of deployment and recovery functions. Additional sub-panels provide display units for mission plan parameters (start location, goal location, and subgoal locations), guidance commands to the AUV (autocourse, autodepth, autospeed), and AUV operating status (xy-coordinates, depth-under-the-auv, heading, depth, and rpm)[5]. The operator is prompted by messages printed to the KEE typescript window for each step of the mission selection and planning process, and all actions performed by the operator are accomplished using the mouse. Also, care has been taken to avoid "information over-load" in the selection and structure of the control panel units and the content of the message prompts. An operator should be able to perform all required functions easily and without guesswork.

This particular version of the control panel is designed to support the simulation systems, NPS AUV-SIM1, 2, and 3, but can be readily modified to support shipboard AUV deployment systems. Each unit on the selection sub-panels are *active images* that activate methods (sub-programs) that perform certain functions that range from simply printing message

[5] The control panel is currently arranged to support the requirements of the NPS AUV simulation systems, but can be easily reconfigured for shipboard use where data feedback may not be available during mission execution.

prompts to executing complex algorithms [Ref. 37]. These methods are in modules that can be easily displayed and edited on the KEE desktop and offer an almost endless range of possibilities for the functionality of the control panel.

An additional feature of the Mission Planning and Control Workstation is the external color monitor that is controlled by the LISP code at the planning level of the software hierarchy. This monitor displays a 2-D representation of the AUV's track during execution of missions using the NPS AUV-SIM2 and 3 simulation systems and the code could be easily modified to display the track of an actual AUV during post-mission analysis. This provides a very nice analysis tool for verifying the correct operation of the AUV.

The experimental results of using the Mission Planning and Control Workstation versus the original mission planning mechanism is not judged in the form of increased speed of operation, but in the form of increased functionality, breadth of information available to the operator, and adaptability to enhanced systems. By extending the use of the KEE expert system shell to develop the Mission Planning and Control Workstation, an improved man-machine interface has been created that greatly contributes to the "laboratory environment" afforded by using the NPS AUV simulation systems and additionally, provides a very adaptable prototype for a shipboard AUV deployment and control system.

a. Analysis of a Typical Mission Profile Test

Numerous tests were conducted during the development of the Mission Planning and Control Workstation to verify the functionality and utility of the system. The NPS AUV-SIM2 simulation system offers the most realistic characteristics of the actual testing environment for the NPS Model 2 AUV and was used in the majority of the testing.

The User's Manual described in the previous chapter offers two methods of loading the knowledge base. Method 1 does not include the preset desktop arrangement and can be a little cumbersome for the first time user, if the proper desktop arrangement has not been left on the screen from a previous mission run. Method 2, though a little slower in some cases, does provide the preset desktop arrangement and is the recommended method of loading the knowledge base for new operators. Once the knowledge base has been loaded and the Deploy icon (located in the AUV-Deploy/Recovery sub-panel) is activated using the mouse, message prompts written in the KEE typescript window guide the operator's every step to set up and run a test mission.

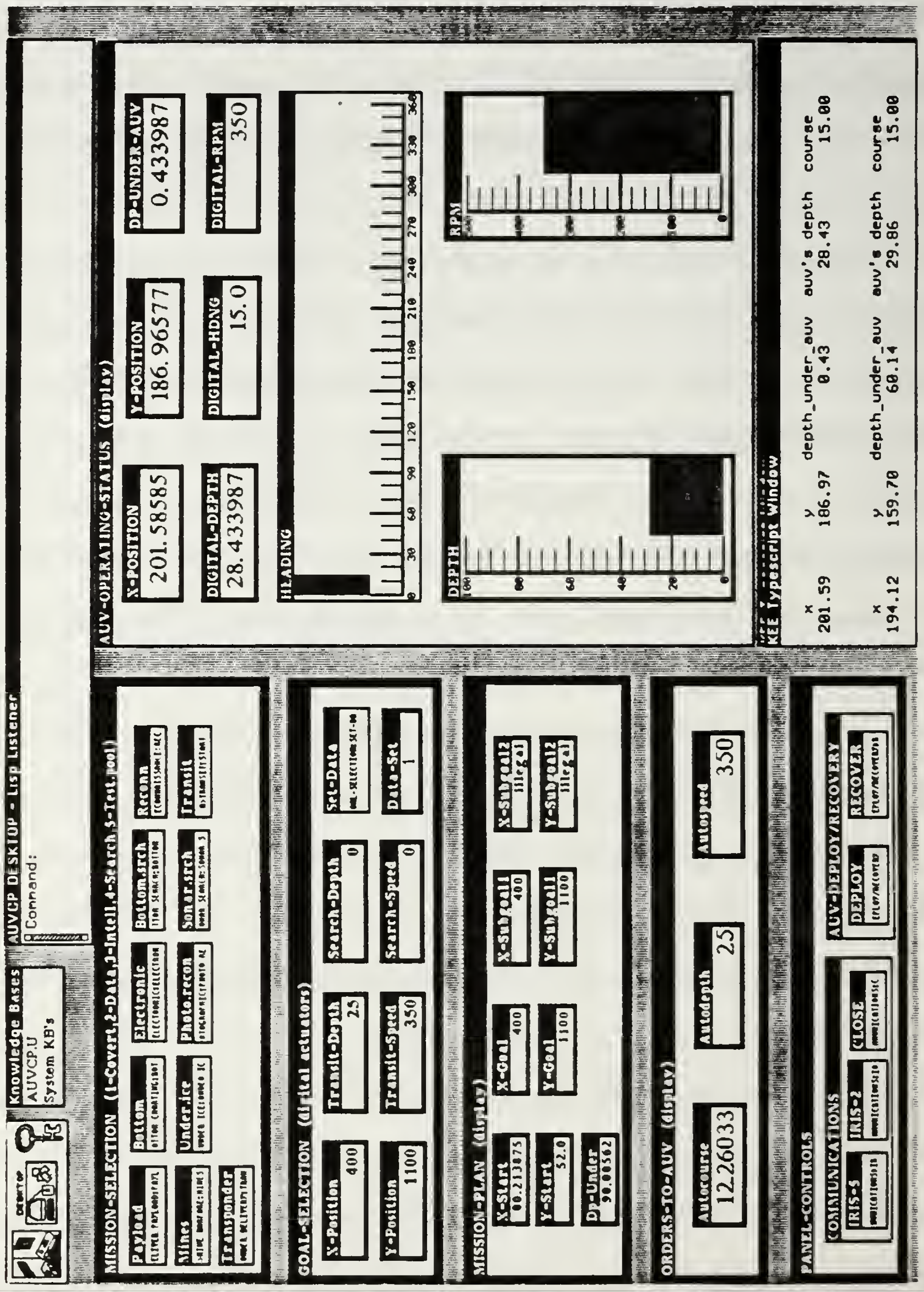
The first step in setting up the simulation system to conduct a test mission run is to load the execution level AUV dynamic model subsystem in the *autopilot* mode. The model can be driven to any desired position within the test pool environment using the *manual* mode and then placed in the *autopilot* mode by pressing the *A-key* on the keyboard of the IRIS machine. For all of the test runs conducted, the AUV model was placed in the lower left corner of the test pool environment and given a goal that would cause the path-planner to plan a diagonal path across the expanse of the test pool that would also pass through the obstacle field. This was done to analyze the time required by the path-planner to plan the particular mission paths through the obstacle field and also to analyze how the AUV model responded to guidance commands issued by the rule-based navigator during the execution of the mission. It was discovered that given a goal (600 1100) that required the AUV to transit almost to the opposite corner of the test pool environment and to cross the approximate middle of the obstacle field, the rule-based path-planner required 7-8 minutes to compute the required path. Conversely, if the goal was selected that required a path that crossed only about one third of the obstacle field (400 1100) the path was computed in less than one minute. In certain real-life scenarios, requiring 7-8 minutes to plan a fairly simple path of this nature might mean the difference between success and failure of the mission. Upgraded path planning algorithms will be required to improve this feature of the simulation system.

The *Transit* mission was selected for all tests, which requires the AUV to be driven to a specified goal location in the test pool environment and then simply turn around and return to its starting location. To support this mission profile, after all mission parameters have been passed to the planning level of the software hierarchy, the path-planner plans the path and causes the start location, goal location, and several subgoal locations to be printed to the KEE typescript window, as well as being displayed on the Mission-Plan sub-panel. If only one subgoal is planned, the *X-Subgoal2* and *Y-Subgoal2* display units in the Mission-Plan sub-panel will show the word *illegal* which tells the operator that the path planner only needs one sub-goal between the start location and the goal location to accomplish the desired transit. For longer, more complex missions, the path-planner will need and will display more subgoals in the

KEE typescript window and the display units in the Mission-Plan sub-panel. For this scenario, as each subgoal is reached, the subgoal locations are updated accordingly in the Mission-Plan sub-panel to reflect the next set of subgoals. The actual goal location will also be displayed as a subgoal once the AUV model nears that location.

Once a mission is executed, guidance commands are immediately transmitted from the planning level rule-based navigator to the execution level dynamic model and the AUV Model begins to move at the commanded speed, turns to the commanded course, and proceeds to the commanded transit depth. Figure 5.1 shows the Mission Planning and Control Panel during a typical test *Transit* mission run with the model in transit toward the designated goal. Its operating status (position, depth, course, speed, and depth-under-the-auv) is displayed in the KEE typescript window and on the AUV-Operating-Status sub-panel of the control panel. Guidance commands are displayed on the Orders-To-AUV sub-panel to allow the operator to readily compare ordered parameters to actual parameters being fed back from the execution level. As shown, the actual heading of the AUV model is 015 as compared to the ordered course of 012.26033 The actual course is accurate to within 81.7% of ordered course, which is acceptable for this level of development.

Figure 5.2 shows the AUV model transiting the test pool environment and Figure 5.3 shows the corresponding AUV track displayed on the external monitor of the Mission Planning and Control Workstation.



X-POSITION 201.58585
Y-POSITION 186.96577
DP-UNDER-AUV 0.433987
DIGITAL-DEPTH 28.433987
DIGITAL-HDNG 15.0
DIGITAL-RPM 350

X-Position 400
Y-Position 1100
Transit-Depth 25
Transit-Speed 350
Search-Depth 0
Search-Speed 0
Set-Data
Data-Set 1

X-Start 80.213875
Y-Start 52.0
Dp-Under 90.00562
X-Goal 400
Y-Goal 1100
X-Subgoal1 400
Y-Subgoal1 1100
X-Subgoal2 illegal
Y-Subgoal2 illegal

Autocourse 12.26033
Autodepth 25
Autospeed 350

```
  x       y      depth_under_auv  auv's depth  course
201.59  186.97        0.43           28.43      15.00

  x       y      depth_under_auv  auv's depth  course
194.12  159.70       60.14           29.86      15.00
```

Figure 5.1 Mission Planning and Control Panel-*Transit* Mission

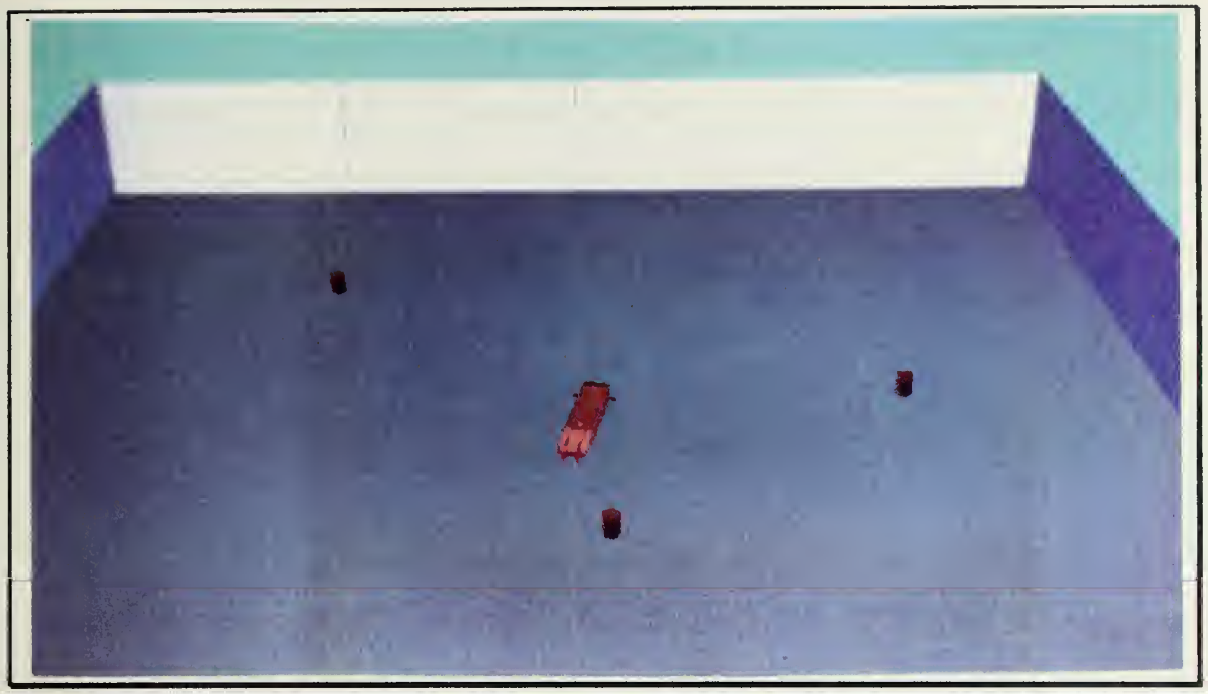


Figure 5.2 AUV Dynamic Model-*Transit* Mission

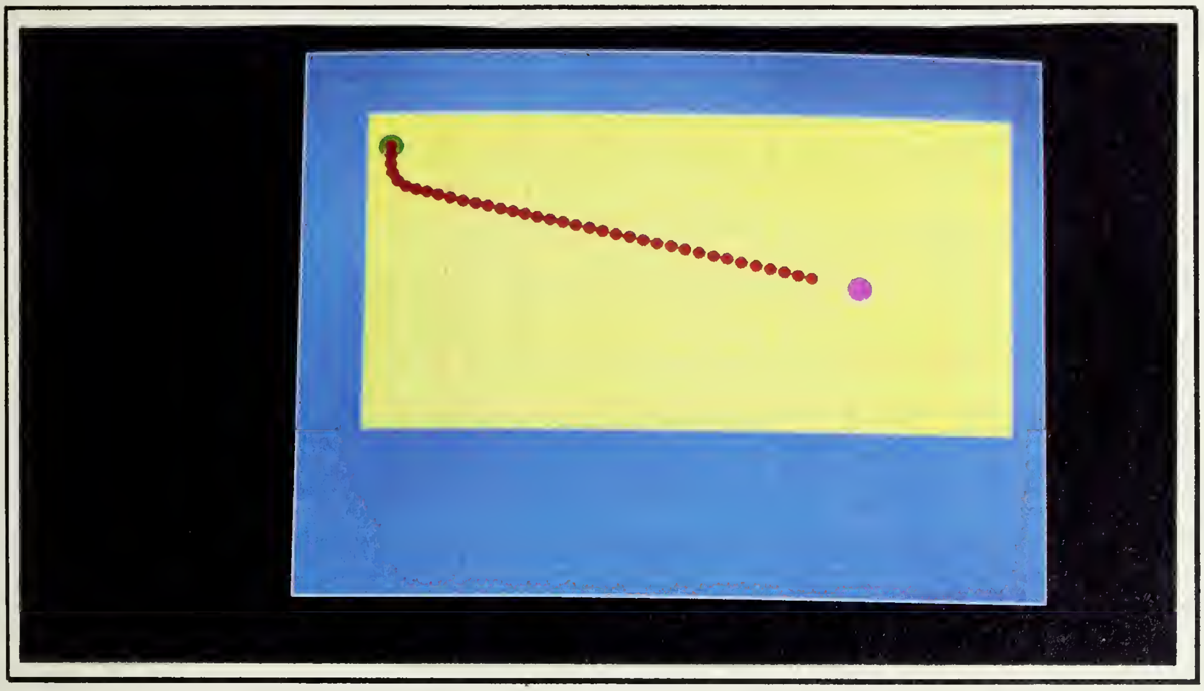
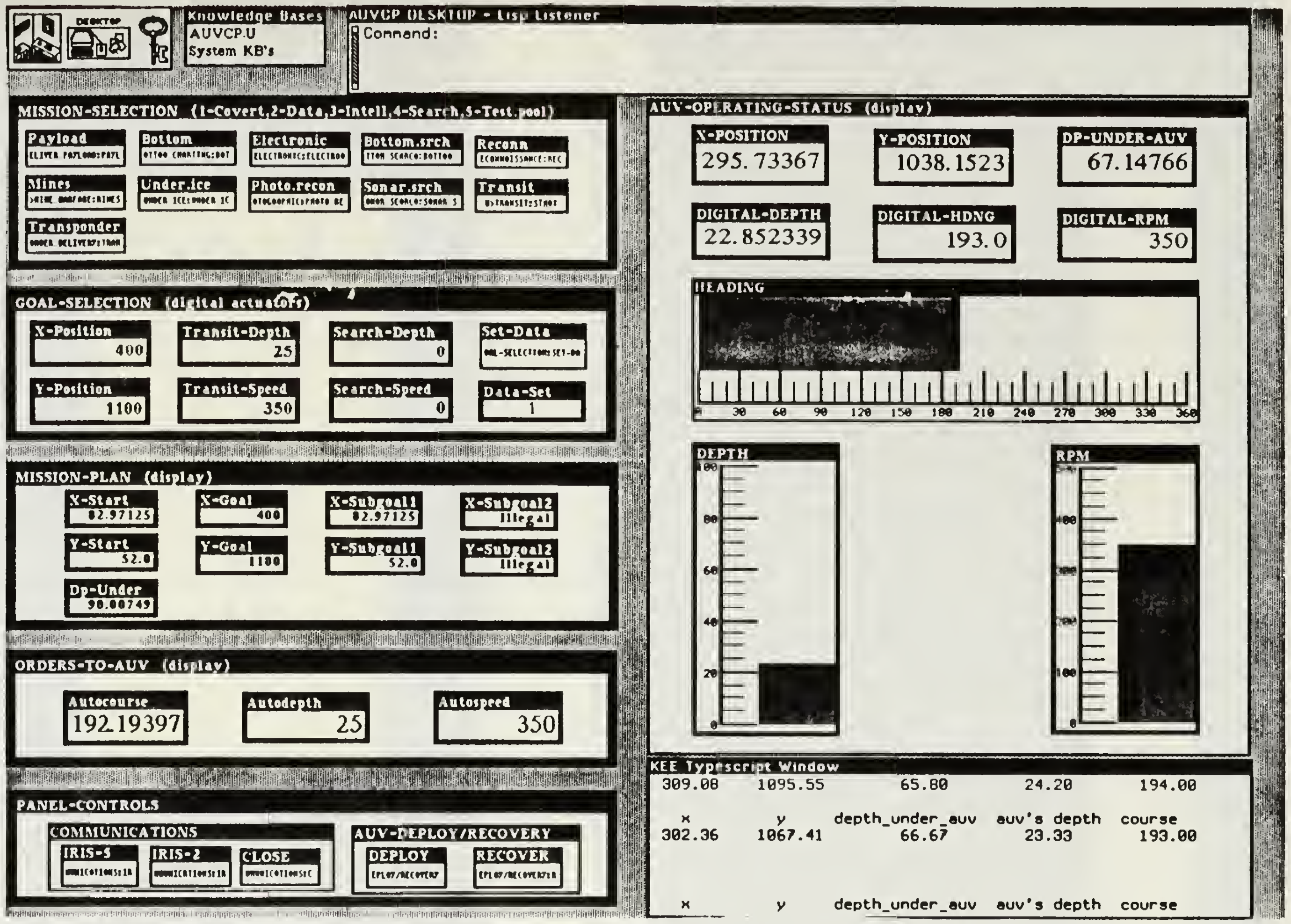


Figure 5.3 Workstation External Monitor-*Transit* Mission

The AUV model's location and attitude correspond to the information displayed for the operator at the Mission Planning and Control Workstation. The dark cylindrical objects in the background are the pre-staged obstacles that have been placed in the test pool environment to test the path planner's ability to plan paths to safely avoid these obstacles. The vertical safety distance is 20 units and the AUV model can be clearly observed changing depth accordingly when it nears any of the obstacles that are within this range of ordered depth.

The external monitor display actually plots the AUV's track during mission execution and provides the operator a ready medium for comparing the AUV's "actual" track to its "ordered" track during post-mission analysis.

Figure 5.4 shows the Mission Planning and Control Panel during the return leg of the *Transit* mission. The AUV model has reached its goal, has reversed course, and is returning to its starting location. Figure 5.5 shows the AUV model in the test pool environment on this return leg of the mission, and Figure 5.6 shows the corresponding AUV model's track on the external monitor portion of the Mission Planning and Control Workstation. Note that the ordered course (autocourse) is 192.129397 on this leg of the transit. The actual course, as shown in the KEE typescript window, is 193.0, which is within 99.5% accuracy when compared to the ordered course. The AUV model appears to be tracking better on the return leg than on the initial leg. The variance on the earlier leg may have been caused by recording the hardcopy of the screen (LISP machine) just


Knowledge Bases
AUVCP.U
System KB's

AUVCP DESKTOP - Lisp Listener
Command:

MISSION-SELECTION (1-Covert,2-Data,3-Intell,4-Search,5-Test.pool)
Payload | Bottom | Electronic | Bottom.srch | Recon
Mines | Under.ice | Photo.recon | Sonar.srch | Transit
Transponder

GOAL-SELECTION (digital actuators)
X-Position 400 | Transit-Depth 25 | Search-Depth 0 | Set-Data
Y-Position 1100 | Transit-Speed 350 | Search-Speed 0 | Data-Set 1

MISSION-PLAN (display)
X-Start 82.97125 | X-Goal 400 | X-Subgoal1 82.97125 | X-Subgoal2 Illegal
Y-Start 52.0 | Y-Goal 1100 | Y-Subgoal1 52.0 | Y-Subgoal2 Illegal
Dp-Under 90.00749

ORDERS-TO-AUV (display)
Autocourse 192.19397 | Autodepth 25 | Autospeed 350

PANEL-CONTROLS
COMMUNICATIONS: IRIS-5 | IRIS-2 | CLOSE
AUV-DEPLOY/RECOVERY: DEPLOY | RECOVER

AUV-OPERATING-STATUS (display)
X-POSITION 295.73367 | Y-POSITION 1038.1523 | DP-UNDER-AUV 67.14766
DIGITAL-DEPTH 22.852339 | DIGITAL-HDNG 193.0 | DIGITAL-RPM 350

KEE Typescript Window
```
 309.08    1095.55        65.80         24.20        194.00

   x          y     depth_under_auv  auv's depth   course
 302.36    1067.41        66.67         23.33        193.00


   x          y     depth_under_auv  auv's depth   course
```

Figure 5.4 Mission Planning and Control Panel-*Transit* Mission

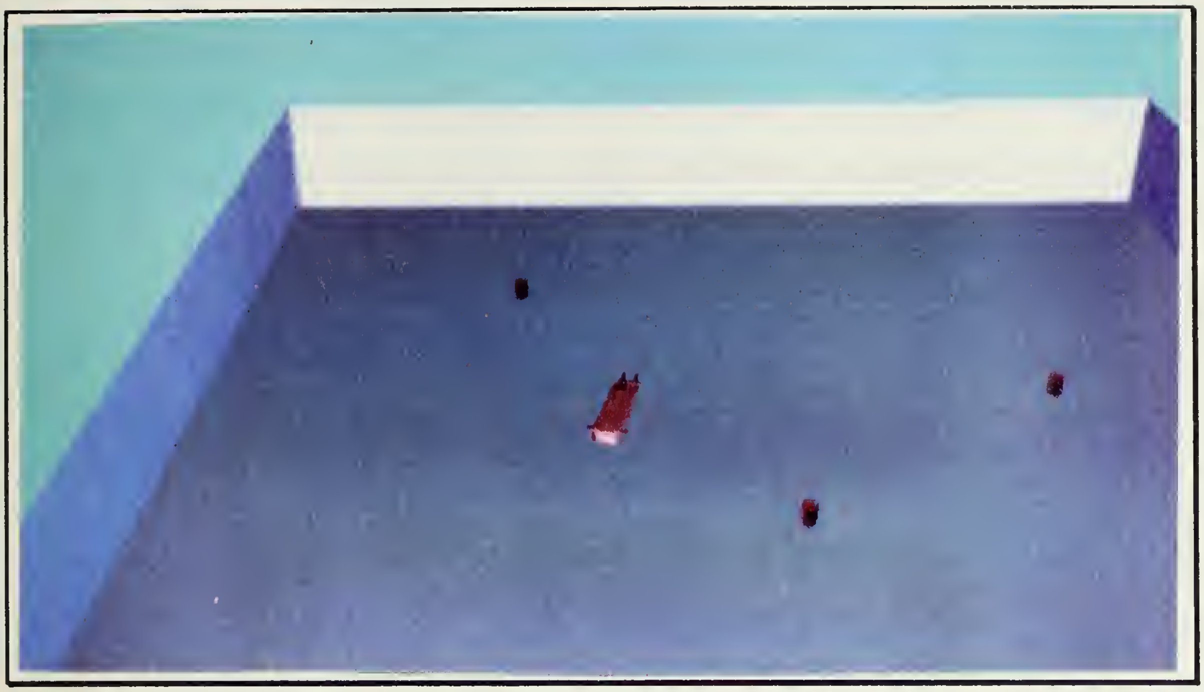


Figure 5.5 AUV Dynamic Model-*Transit* Mission

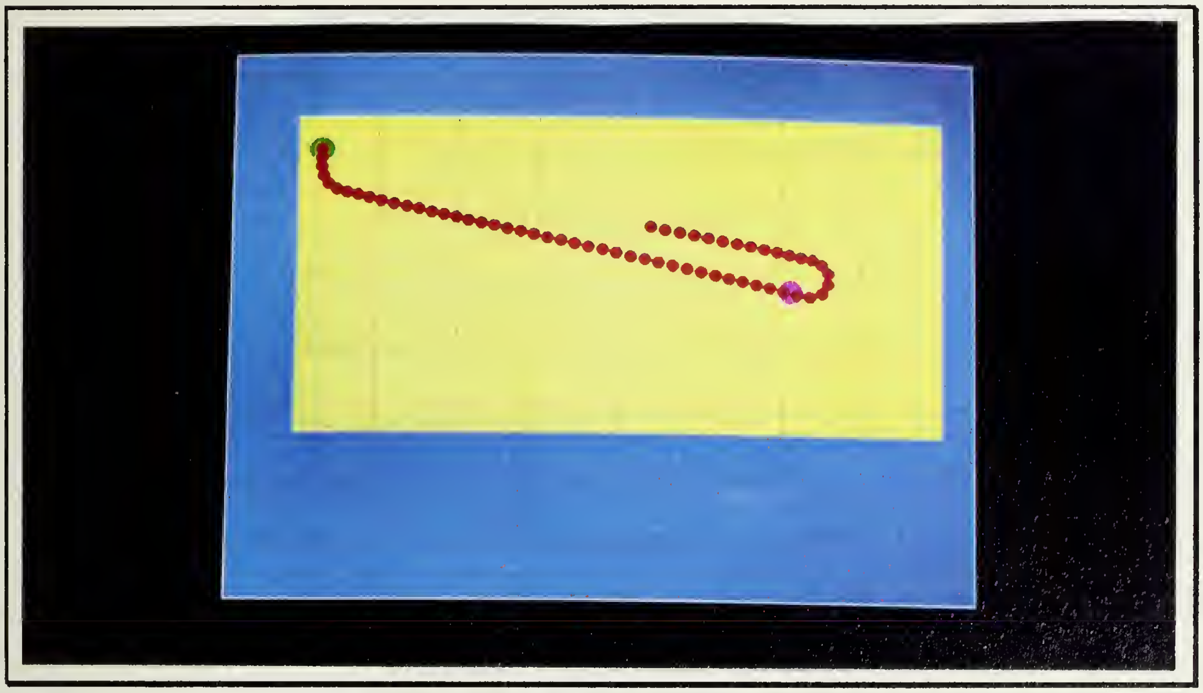


Figure 5.6 Workstation External Monitor-*Transit* Mission

after a slight course change, in which case, the AUV model would still have been turning to ordered course. All other indications show that the AUV model was following ordered guidance commands accurately. As a case in point, Figure 5.6 shows a fairly straight line track to the goal location and that the AUV model passed through the middle of this location (designated by the magenta circle on the grid) before reversing course to start the return leg of the mission. This substantiates the earlier assumption that the AUV model followed ordered guidance commands closely.

Figure 5.6 also shows that the AUV model, though it clearly goes back to its starting location as desired, does not return to its original track during its return transit. This occurs on this test run because the path planner only provided one subgoal between the starting position and the goal. These same positions (waypoints) are also used for the return transit but in the reverse order. Additionally, the algorithm that designates the return transit route for the AUV model deletes the goal position from the list of waypoints so as not to confuse the AUV model. When the AUV model passes through the goal position and reverses course, if the goal is in the list of waypoints that it must pass through, it is too close to this position to pass through the point and also maintain the ordered course and will begin an infinite circle around this point. Clearly, the algorithm must be improved to avoid this problem. Subsequent testing with longer transit paths and more waypoints between the starting location and the goal revealed that the AUV model does angle back to its original track after passing through the goal location and reversing course. This is the desired result, but should occur immediately following the course reversal to avoid obstacles that may lie close to its return track.

The test results shown and described here clearly show the functionality of the Mission Planing and Control Workstation and how well the AUV dynamic model responds to planning level guidance commands. Needed improvements to the path planning algorithm have been identified and should be implemented during future research.

3. Missions Upgrade

The *Transit* mission and supporting functions of the Test.Pool category were modified to support the various functions of the Mission Planning and Control Workstation. Currently, this is the only category of mission profiles that support controlling the AUV dynamic model in the test pool environment. The other missions support the open-ocean format NPS AUV-SIM1 simulation system. The modifications to this code provide an excellent framework for upgrading more complex mission profiles to control the AUV dynamic model and will be the subject of future research.

4. Mission Planner/Dynamic Model Interface

Currently, the AUV dynamic model included in the NPS AUV-SIM3 simulation system is primarily used for testing various maneuvering subsystem (control surfaces) controller algorithms. The software structure (C code) of the dynamic model for this simulation system is arranged in a modular format, which makes it an excellent vehicle for testing different software subsystems. This dynamic model was integrated into a complete simulation system, very similar to the NPS AUV-SIM2 system, to allow this model to be controlled in the autopilot mode and be fully tested while running mission profiles. Though the test pool environment is not to scale, it should be a fairly simple modification to convert the water environment's dimensions to conform to those of the pool where the NPS Model 2 AUV will be tested. Additionally, the modular format of this model allows the external configuration to be easily modified to look and perform more like the real vehicle once its characteristics have been more fully defined.

The LISP-IRIS machine interface code has also been put into a modular format. This allows the integration of any new dynamic models into the simulation system with much more ease than earlier formats. The new interface code can be developed outside the main body of the graphics package, and modules can be simply swapped for testing of the new model.

C. SUMMARY

This chapter presents a summary of the upgrades to the NPS AUV simulation systems. The functionality and utility of the Mission Planning and Control Workstation is contrasted to the earlier format and a typical test of a mission profile is thoroughly analyzed. Next, upgrades to the Test Pool mission profiles to support this workstation are discussed. Lastly, the integration of the latest version of the AUV dynamic model into a complete simulation system that affords both manual and autopilot operation is discussed along with, upgrades to the LISP-IRIS machine interface.

VI. SUMMARY AND CONCLUSIONS

A. RESEARCH CONTRIBUTIONS

As originally reported by Nordman [Ref. 9], the NPS AUV simulator systems are important tools for incorporating new control concepts and algorithms into the latest version of the NPS AUV. In addition, the KEE expert system shell has facilitated the development of a Mission Planning and Control Workstation that further extends the development possibilities of these simulation systems. At the top level of the software control structure, this new utility provides an operator with an informative, easy-to-operate, and totally interactive control panel that allows rapid mission planning and execution of the NPS AUV simulation systems. In addition, the control panel displays real time feedback data for immediate analysis of the operation of the AUV dynamic model. The control panel is a prototype for use aboard AUV deployment platforms and can be easily configured to display pre-deployment readiness status, real time feedback display, or post mission analysis display from a data file generated by the AUV's onboard computer. The system also retains the flexibility to allow programmers to readily modify or develop new mission and planning level code and mission profiles.

This enhanced "laboratory environment" gives the operator more of a feel of real-life operations and is a valuable test and debugging tool that will save countless hours of experimentation and allow rapid verification of subsystem software code reliability before installation into the actual vehicle.

B. RESEARCH EXTENSIONS

Several research extensions of NPS AUV simulation system development, discussed by both MacPherson [Ref. 8] and Nordman [Ref. 9], still apply and must be re-iterated here. These extensions include developing faster path-planning algorithms, an AUV vision system for mapping and contact classification, an environment for examining high resolution sonar data, a hovering mode with appropriate guidance command control, and incorporation of a simple and user-friendly interface that allows selection of different dynamic models for rapid comparative analysis. The current state of the "laboratory environment" afforded by the existing NPS AUV simulation systems make these extensions not only reasonable, but fairly easily developed and implemented.

Currently, research conducted by Friend [Ref. 38], has made available a navigation subsystem algorithm and a corresponding 3-D graphics simulation system using simulated inputs from the sonar transducers proposed for the NPS Model 2 AUV. This simulation system should be integrated into the NPS AUV-SIM2 and 3 simulation systems for further analysis and development of sonar subsystem interface software for the actual vehicle.

A research proposal and plan is currently being prepared to develop a series of path-planning algorithms that will be specifically tailored for different real-life scenarios (i.e. quick-reaction missions, long-range missions, close-quarters missions, etc). These algorithms should be readily selectable from the Mission Planning and Control Panel and should utilize the KEE rules language to develop this interface.

Current plans for the NPS Model 2 AUV include having two onboard computers. One of the computers (386-based MS-DOS gridcase) will be used for data storage and to possibly house real-time planning level LISP code, and the second computer (GESPAC, Motorola-based, 68020, with OS-9 operating system) will house the primary autopilot control code, written in C or possibly ADA. To fully develop and test the software code for this system, the simulation systems should be upgraded to include a simulated three computer system by interconnecting a gridcase computer between the LISP machine and the IRIS graphics workstation. Additionally, research must be conducted to ascertain the best format for the software structure of the onboard computer systems and the format of the information that is to be downloaded from the mission level to these systems. This information could be in the form of mission parameters, a set of mission rules, or a combination of both. The onboard systems could accept this information and either perform all of the planning level functions within its software structure or contain a rule system that would allow it to only modify mission plans downloaded to it based on real-time situations. Upgrading the simulation systems would allow this research to be conducted rapidly and concurrently with the construction of the actual vehicle.

The NPS AUV-SIM3 dynamic model is being used to develop various propulsion and maneuvering subsystem controller software. The modular software structure of this simulation system makes this an ideal vehicle for this research. The simulation system should, however, be frequently updated with the new algorithms and thoroughly tested, both in the manual and autopilot modes. Additionally, as the physical design of the NPS Model 2 AUV is more fully defined, a new dynamic model should be developed to research probable operating characteristics of the new design. As testing of the physical vehicle begins, this dynamic model should be updated frequently with the most current hydrodynamic data available. These simulation systems can provide a highly realistic simulation of the actual vehicle's operating characteristics and, if utilized to its fullest potential, will produce valuable and timely feedback by quickly demonstrating the validity of or the potential problems/side effects created by the new design.

A fully operational and versatile AUV should have the ability to deviate from planned mission profiles for obstacle avoidance, mapping of unexpected/ interesting objects, etc. and then be able to resume the pre-planned mission or re-plan the mission based on new path information. Additionally, the AUV should be "smart" enough to "learn" from encounters with previously unknown obstacles so as to plan better paths for future missions.

Nordman [Ref. 9] suggested that the KEE expert system shell's rule language could be utilized to implement "interrupt" and "hover" decision based code to facilitate obstacle avoidance/exploration behavior by the

AUV. KEE provides an excellent rule system that provides an effective way to: 1) express information in discrete steps as independent pieces or modules of knowledge, 2) express unordered and declarative information, and 3) specify conditions under which an action takes place [Ref. 37]. By expressing information in rules, the knowledge base remains transparent, easily readable, and modular. These characteristics facilitate ready development of applications like obstacle avoidance/exploration as well as scenario-driven mission and path planning. The current knowledge base should be further extended to make use of this facility while retaining the Mission Planning and Control Panel format.

Currently, all of the mission profiles, with the exception of the Test Pool category, are written for the open-ocean simulator system, NPS AUV-SIM1. These missions should be upgraded to facilitate them being run in the test pool environment. This is in preparation for testing of the actual vehicle in its water environment. As the characteristics of the NPS Model 2 AUV become more well defined, and the simulation systems are upgraded to reflect these operating characteristics, more detailed testing can be done with the more sophisticated missions to determine inadequacies or side-effects of the new design that might not make it practical for certain scenarios/situations. Changes to the design could be made accordingly. This will provide an excellent medium for validating the design as it is being developed, and should facilitate an overall time and cost savings in the full development of the NPS AUV.

APPENDIX

This appendix contains files of computer code that is pertinent to this thesis research. The two major knowledge base files, *auvcp.u* and *auvcp.desktop*, are generated automatically by the KEE system as the knowledge base is created (active images, image panels, frames, etc). These files are not included as they are very long and complex and would not practically benefit the reader of this report. The files that are included are listed below with brief explanations:

* File *ap3.lisp*: File *ap3.lisp* gets mission goal position from the KEE package and initial AUV position info from the IRIS machine and calculates the autopilot course for the simulator to steer during mission execution.

* File *initkb.lisp*: File *initkb.lisp* initializes the display components of the AUV Mission Planning and Control Panel. At the start of each mission selection process, all display units are set to zero.

* File *monitor.lisp*: File *monitor.lisp* creates a color display screen on the LISP machine external color monitor. The display screen contains a 2-D representation of the test pool environment for the AUV dynamic model. During mission execution, AUV position data is sent to this file from from file *ap3.lisp* and the AUV's track is plotted using a colored icon.

* File *best.lisp*: File *best.lisp* was created during earlier research [Ref. 8] and is provided here for continuity. This lisp code accepts start/goal positions from ap3.lisp and uses a best-first search algorithm to calculate the appropriate path from start to goal.

* File *sym-iris-comm1.lisp*: This file is also provided for continuity. It is an updated version of *sym-iris-comm.lisp* [Ref. 9] developed by Professor S. Kwak. This file facilitates the interprocess communications between LISP machines and IRIS graphics workstations for the NPS AUV-SIM2 and 3 simulation systems.

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER. Base: 10 -*-


; This is the file ap3. This lisp code gets mission goal position from the KEE package
; and initial AUV position info from the IRIS machine and calculates the autopilot course
; for the simulator to steer during mission execution.

; The following missions are designed for the submarine simulator currently
; running on IRIS3. These are open ocean missions an AUV might be expected
; to carry out.

(defun elect_recon_mission (xdest ydest time_on_station)
   (transit_to_pt xdest ydest 200 5)
   (come_to_PD xdest ydest)
   (record_data_on_station xdest ydest time_on_station)
   (transit_back 200 5)
   (come_to_PD xstart ystart)
   (princ "ELECTRONIC MISSION COMPLETED"))

(defun photo_recon_mission (xdest ydest time_on_station periscope_bearing)
   (transit_to_pt xdest ydest 300 6)
   (come_to_PD xdest ydest)
   (take_photos_on_station xdest ydest time_on_station periscope_bearing)
   (transit_back 300 6)
   (come_to_PD xstart ystart)
   (princ "PHOTO RECON MISSION COMPLETED")
   (standby_for_recovery xstart ystart))


(defun sonar_search_mission (xdest ydest search_depth search_speed)
   (transit_to_pt xdest ydest 300 5)
   (execute_sonar_search xdest ydest search_depth search_speed)
   (transit_back 300 5)
   (come_to_PD xstart ystart)
   (princ "SONAR SEARCH MISSION COMPLETED"))

(defun execute_sonar_search (xdest ydest search_depth search_speed)
   (sonar_search (- xdest 30) (+ 5 ydest) search_depth search_speed )
   (sonar_search (+ xdest 30) (+ 10 ydest) search_depth search_speed )
   (sonar_search (- xdest 30) (+ 15 ydest) search_depth search_speed )
   (sonar_search (+ xdest 30) (+ 20 ydest) search_depth search_speed )
   (princ "SONAR SEARCH COMPLETED"))

(defun sonar_search (xsearch ysearch depth speed)
  (do ((distance_to_goal (get_the_distance x y xsearch ysearch)
                    (get_the_distance x y xsearch ysearch)))
     ((> 2 distance_to_goal) (princ "SUB AT SUBGOAL"))
     (send_float (get_autocourse x y xsearch ysearch))
     (send_float depth)
     (send_float speed)
     (send_float xsearch)
     (send_float ysearch)
     (send_string "sonar search")
     (get_data_from_iris)))
```

```
(defun bottom_search_mission (xdest ydest search_speed)
   (transit_to_pt xdest ydest 300 4)
   (dive_to_bottom xdest ydest)
   (execute_bottom_search xdest ydest search_speed)
   (transit_back 300 4)
   (come_to_PD xstart ystart)
   (princ "BOTTOM SEARCH MISSION COMPLETED"))

(defun dive_to_bottom (xdest ydest)
   (do ((depth_now sub_depth sub_depth))
       ((< depth_under_sub 150) (princ "SUB NEAR BOTTOM"))
        (send_float (get_autocourse x y xdest ydest))
        (send_float (- (+ sub_depth depth_under_sub) 145))
        (send_float 4)
        (send_float x)
        (send_float y)
        (send_string "DIVE TO THE BOTTOM")
        (get_data_from_iris)))

(defun execute_bottom_search (xdest ydest search_speed)
   (bottom_search (- xdest 30) (+ ydest 5) search_speed)
   (bottom_search (+ xdest 30) (+ ydest 10) search_speed)
   (bottom_search (- xdest 30) (+ ydest 15) search_speed)
   (bottom_search (+ xdest 30) (+ ydest 20) search_speed))

(defun bottom_search (xsearch ysearch search_speed)
   (do ((distance_to_goal (get_the_distance x y xsearch ysearch)
                          (get_the_distance x y xsearch ysearch)))
       ((> 2 distance_to_goal) (princ "SUB AT SUBGOAL"))
        (send_float (get_autocourse x y xsearch ysearch))
        (send_float (- (+ sub_depth depth_under_sub) 150))
        (send_float search_speed)
        (send_float xsearch)
        (send_float ysearch)
        (send_string "BOTTOM SEARCH")
        (get_data_from_iris)))


(defun deliver_payload_mission (xdest ydest transit_depth transit_speed)
   (transit_to_pt xdest ydest transit_depth transit_speed)
   (come_to_PD xdest ydest)
   (princ "TRANSIT COMPLETED - RECOVER PAYLOAD")
   (standby_for_recovery xdest ydest))

;;; The following functions support driving the AUV dynamic model on IRIS5 in the
;;; "autopilot mode". They are based on the "deliver_payload_mission" and "transit_to_pt"
;;; functions and do not yet support collision avoidance. They do handle path-planning
;;; around obstacles. Any expressions that start with "kee::..." designate KEE package
;;; functions that pass info from this, the USER's package to the KEE package for display on
;;; the Mission Planning and Control Panel.

(defun transit_pool (x1 y1 transit_depth transit_speed)
   (start-con)
   (terpri)
```

```
    (princ "Connection with iris established.")
    (terpri)
    (setq xstart (get_data))
    (setq x xstart)
    (princ "x received from iris:  ") (print x)
    (terpri)
    (kee::put.value 'kee::mission-plan 'kee::x-start xstart)
    (kee::put.value 'kee::auv-operating-status 'kee::x-position xstart)
    (setq ystart (get_data))
    (setq y ystart)
    (princ "y received from iris:  ") (print y)
    (terpri)
    (kee::put.value 'kee::mission-plan 'kee::y-start ystart)
    (kee::put.value 'kee::auv-operating-status 'kee::y-position ystart)
    (setq depth_under_sub (get_data))
    (princ "depth_under_sub received from iris:  ")(prin1 depth_under_sub)
    (terpri)
    (kee::put.value 'kee::mission-plan 'kee::depth-under-auv depth_under_sub)
    (kee::put.value 'kee::auv-operating-status 'kee::depth-under-auv depth_under_sub)
    ;;; The following lines of code send data to the color monitor for display of the
    ;;; AUV model's start position and the goal position.
    ;;; The color monitor's coord system is opposite that of the iris.
    ;;; (x direction on iris = y direction on monitor)
    (setq xi ystart)
    (setq yi xstart)
    (setq xg y1)
    (setq yg x1)
    (draw-start-pos xi yi)
    (draw-goal-pos xg yg)
    (move-icon xi yi)
    ;;;
    (plan_path x y x1 y1 transit_depth)
    (setq rev_path (reverse path))
    (terpri)
    (princ "Autopilot course calculated for first leg.")(terpri)
    (princ "Hit a key on Iris5 main terminal to continue.")(terpri)
    (send_float x1)
    (send_float y1)
    (transit_without_contacts x1 y1 transit_depth transit_speed "TRANSIT")
    (transit_back_without_contacts transit_depth transit_speed)
    (terpri)
    (princ "TRANSIT COMPLETE.")
    (terpri)
    (stop_in_pool xstart ystart))


; The following functions support the AUV and SUB simulator functions in the
; above code. These are subsidiary functions carried out during a mission, or
; functions that assist the path planner with course and water depth calculations.


(defun transit_back (autodepth autospeed)
    (setq path rev_path)
    (setq path (cdr path))       ;drop the first node to prevent auv limit cycling...
    (transit xstart xstart autodepth autospeed "TRANSIT BACK"))


(defun record_data_on_station (xdest ydest minutes)
    (setq end_time (+ (* minutes 3600) (get-internal-real-time)))
    (do ((time_now (get-internal-real-time)
                (get-internal-real-time)))
        ((> time_now end_time) (princ "OK"))
        (send_float (get_autocourse x y xdest ydest))
        (send_float 0)
        (send_float 2)
        (send_float x)
        (send_float y)
        (setq command "ELECTRONIC RECON - ANTENNA RAISED")
        (send_string command)
        (get_data_from_iris)))

(defun take_photos_on_station (xdest ydest minutes periscope_bearing)
    (setq end_time (+ (* minutes 3600) (get-internal-real-time)))
    (do ((time_now (get-internal-real-time)
                (get-internal-real-time)))
        ((> time_now end_time) (princ "OK"))
        (send_float (get_autocourse x y xdest ydest))
        (send_float 0)
        (send_float 2)
        (send_float x)
        (send_float y)
        (send_string "PHOTOGRAPHIC RECON - PERISCOPE TRAINED")
        (get_data_from_iris)))

(defun come_to_PD (xdest ydest)
   (do ((depth_now sub_depth sub_depth))
      ((< depth_now 1) (princ "SUB at PD"))
        (send_float (get_autocourse x y xdest ydest))  ;send autocourse
        (send_float 0)  ;send autodepth
        (send_float 4)  ;send autospeed
        (send_float x)
        (send_float y)
        (send_string "COME TO PERISCOPE DEPTH")
        (get_data_from_iris)))

(defun standby_for_recovery (xdest ydest)
   (loop
        (send_float (get_autocourse x y xdest ydest))
        (send_float 0)  ;send autodepth
        (send_float 2)  ;send autospeed
        (send_float xdest)
        (send_float ydest)
        (send_string "STANDING BY FOR RECOVERY - ANTENNA RAISED")
        (get_data_from_iris)))

(defun get_data_from_iris()
        (setq x (get_data))
        (setq y (get_data))
        (setq depth_under_sub (get_data))
        (setq sub_depth (get_data))
        (setq acourse (get_data))
        (princ "   x              y          depth_under_sub    sub's depth      course")
        (format t "~% ~0.2F ~15.2F ~15.2f ~15.2F ~15.2F" x y depth_under_sub sub_depth acourse)
        (terpri)
        (setq sonar_contacts
              (list
                 (list (get_data) (get_data))
                 (list (get_data) (get_data))
                 (list (get_data) (get_data))
                 (list (get_data) (get_data))
                 (list (get_data) (get_data))
                 (list (get_data) (get_data))
                 (list (get_data) (get_data))
                 (list (get_data) (get_data))
                 (list (get_data) (get_data))))
;         (princ sonar_contacts)
              (princ "Contact ranges:")(terpri)
        (get_closest_range sonar_contacts)
        (terpri))

(defun get_data_from_iris_without_contacts()
        (setq x (get_data))
        (kee::put.value 'kee::auv-operating-status 'kee::x-position x)
              (setq y (get_data))
        (kee::put.value 'kee::auv-operating-status 'kee::y-position y)
              ;;; The following few lines transfer data to the color monitor to mark the AUV's
              ;;; track during mission execution.
              ;;; The color monitor coord system is opposite that of iris display;
              ;;; (x direction on iris = y direction on monitor)
              (setq xi y)
              (setq yi x)
        (move-icon xi yi)
              ;;;
              (setq depth_under_sub (get_data))
        (kee::put.value 'kee::auv-operating-status 'kee::depth-under-auv depth_under_sub)
              (setq sub_depth (get_data))
        (kee::put.value 'kee::auv-operating-status 'kee::auv-depth sub_depth)
              (setq acourse (get_data))
        (kee::put.value 'kee::auv-operating-status 'kee::heading acourse)
              (princ "   x         y     depth_under_auv  auv's depth  course")
              (format t "~% ~0.2F ~10.2F ~12.2f ~12.2F ~12.2F" x y depth_under_sub sub_depth acourse)
              (terpri))


(setq closest_contact_range 3)


(defun get_closest_range (contacts)
    (do ((contact_list contacts
                (cdr contact_list))
        (list_length (length contacts)
                (1- list_length)))
        ((< list_length 2))
        (princ (caar contact_list))
        (terpri)))

(defun transit_to_pt (x1 y1 autodepth autospeed)
    (start-con)
    (terpri)
    (princ "Connection with iris established.")
    (terpri)
    (setq xstart (get_data))
    (setq x xstart)
    (princ "x received from iris: ") (print x)
    (terpri)
    (setq ystart (get_data))
    (setq y ystart)
    (princ "y received from iris: ") (print y)
    (terpri)
    (setq depth_under_sub (get_data))
    (princ "depth_under_sub received from iris: ") (prin1 depth_under_sub)
    (send_float x1)
    (send_float y1)
    (plan_path x y x1 y1 autodepth)
    (setq rev_path (reverse path))
    (terpri)
    (princ "Autopilot course calculated for first leg.")(terpri)
    (princ "Hit <Enter> on Iris2 side terminal to continue.")
    (terpri)
    (transit x1 y1 autodepth autospeed "transit"))

(defun transit (x1 y1 autodepth autospeed sub_command)
    (do ((distance_to_goal (get_the_distance x y x1 y1)
                     (get_the_distance x y x1 y1)))
      ((> 2 distance_to_goal) (princ "SUB AT GOAL"))
     (setq autocourse (get_autocourse x y (caadr path) (cadadr path)))
     (send_float autocourse)
     (cond
       ((> 100 depth_under_sub) (send_float (- (+ depth_under_sub sub_depth) 100)))
       (t (send_float autodepth)))
     (send_float autospeed)
     (send_float (caadr path))
     (send_float (cadadr path))
     (send_string sub_command)
          (terpri)
     (get_data_from_iris)
          (cond
            ((> 1 (get_the_distance  x y (caadr path) (cadadr path)))
             (setq path (cdr path) )   ))))

(defun transit_without_contacts (x1 y1 autodepth autospeed sub_command)
     (do ((distance_to_goal (get_the_distance x y x1 y1)
                                    (get_the_distance x y x1 y1)))
      ((> 25 distance_to_goal) (terpri) (princ "AUV AT GOAL") (terpri))
     (setq autocourse (get_autocourse x y (caadr path) (cadadr path)))
          (send_float autocourse)
     (kee::put.value 'kee::orders-to-auv 'kee::autocourse autocourse)
          (cond
            ((> 20 depth_under_sub) (send_float (- (+ depth_under_sub sub_depth) 20)))
```

```
              (t (send_float autodepth)))
        (kee::put.value 'kee::orders-to-auv 'kee::autodepth autodepth)
              (send_float autospeed)
        (kee::put.value 'kee::orders-to-auv 'kee::autospeed autospeed)
        (kee::put.value 'kee::auv-operating-status 'kee::auv-rpm autospeed)
              (send_float (caadr path))
        (kee::put.value 'kee::mission-plan 'kee::x-subgoal1 (caadr path))
              (send_float (cadadr path))
        (kee::put.value 'kee::mission-plan 'kee::y-subgoal1 (cadadr path))
        (kee::put.value 'kee::mission-plan 'kee::x-subgoal2 (caadr (cdr path)))
        (kee::put.value 'kee::mission-plan 'kee::y-subgoal2 (cadadr (cdr path)))
              (send_string sub_command)
              (terpri)
              (get_data_from_iris_without_contacts)
              (cond
                ((> 25 (get_the_distance x y (caadr path) (cadadr path)))
                 (setq path (cdr path)  )))))

(defun transit_back_without_contacts (transit_depth transit_speed)
  (setq path rev_path)
  (transit_without_contacts xstart ystart transit_depth transit_speed "TRANSIT BACK"))

(defun stop_in_pool (xstart ystart)
  (loop
     (send_float (get_autocourse x y xstart ystart))
     (send_float 0)                          ; put auv on surface.
     (send_float 0)                          ; come to all stop.
     (send_float xstart)
     (send_float ystart)
     (kee::put.value 'kee::auv-operating-status 'kee::auv-rpm 0)
     (send_string "STANDING BY FOR RECOVERY.")
     (get_data_from_iris_without_contacts)))

(defun plan_path (x y x1 y1 autodepth)
  (if
   (null (check_water_depth_along_track x y x1 y1 autodepth))
   (setq path (process_path (get_real_path (list x y) (list x1 y1) (+ 100 autodepth) 20)
                                             (+ 100 autodepth)))
   (setq path (list (list x y) (list x1 y1))))
  (prin1 path))


(defun get_autocourse (x y x1 y1 )
  (cond
   ((< x x1) (autocourse1 x y x1 y1))
   (t (- 360 (autocourse1 x y x1 y1)))))

(defun autocourse1 (x y x1 y1)
  (* 57.295 (acos (/ ( - y1 y)
                (get_the_distance x y x1 y1)))))


(defun get_the_distance (x y x1 y1)
  (sqrt (+ (square (- x x1))
                (square (- y y1)))))



(defun check_water_depth_btwn_nodes (n1 n2 autodepth)
 (check_water_depth_along_track (car n1) (cadr n1)
                                         (car n2) (cadr n2) autodepth))

(defun check_water_depth_along_track (x y x1 y1 autodepth)
  (setq ac (get_autocourse x y x1 y1))
  (setq track_length (get_the_distance x y x1 y1))
  (setq xx x)
  (setq yy y)
  (prog ((index track_length))
   again
         (cond ((> 0 index) (return index)))
         (setq index (1- index))
     (setq xx (+ xx (sin (/ ac 57.295))))
     (setq yy (+ yy (cos (/ ac 57.295))))
;     (prin1 (get_water_depth xx yy)) (princ "  ")
;     (prin1 xx) (princ "   ") (prin1 yy) (terpri)
         (if (< autodepth (get_water_depth xx yy))
            (go again) nil))
  (if
         (> 1 (get_the_distance xx yy x1 y1))
     (princ " --SUFFICIENT WATER DEPTH ALONG INTENDED TRACK-- ")))

(defun process_path (path autodepth)
 (cond
  ((and (< 2 (length path))
      (check_water_depth_btwn_nodes (car path) (cadr path) autodepth))
   (process_path (cons (car path) (cddr path)) autodepth))

  ((equal 2 (length path)) path)
  (t (setq path (cons (car path)
                              (process_path (cdr path) autodepth))))))
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: KEE; Base: 10 -*-

(in-package 'kee)

;; This is file initkb. This file initializes the display components
;; of the AUV Mission Planning and Control Panel. At the start of each
;; mission selection process all display units are set to zero.

(defun init-displays ()
;;initialize goal-selection parameters
 (put.value 'goal-selection 'x-position 0)
 (put.value 'goal-selection 'y-position 0)
 (put.value 'goal-selection 'transit-depth 0)
 (put.value 'goal-selection 'transit-speed 0)
 (put.value 'goal-selection 'search-depth 0)
 (put.value 'goal-selection 'search-speed 0)
 (put.value 'goal-selection 'data-set 0)

;;initialize mission-plan parameters

 (put.value 'mission-plan 'x-start 0)
 (put.value 'mission-plan 'y-start 0)
 (put.value 'mission-plan 'depth-under-auv 0)
 (put.value 'mission-plan 'x-goal 0)
 (put.value 'mission-plan 'y-goal 0)
 (put.value 'mission-plan 'x-subgoal1 0)
 (put.value 'mission-plan 'y-subgoal1 0)
 (put.value 'mission-plan 'x-subgoal2 0)
 (put.value 'mission-plan 'y-subgoal2 0)

;;initialize orders-to-auv parameters

 (put.value 'orders-to-auv 'autocourse 0)
 (put.value 'orders-to-auv 'autodepth 0)
 (put.value 'orders-to-auv 'autospeed 0)

;;initialize auv-operating-status

 (put.value 'auv-operating-status 'x-position 0)
 (put.value 'auv-operating-status 'y-position 0)
 (put.value 'auv-operating-status 'depth-under-auv 0)
 (put.value 'auv-operating-status 'auv-depth 0)
 (put.value 'auv-operating-status 'auv-rpm 0)
 (put.value 'auv-operating-status 'heading 0))

(defun setpack-user ()
;;sets the user package
 (setf *package* (find-package "user")))

(defun setpack-kee ()
;;sets the kee package
 (setf *package* (find-package "kee")))
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER -*-

;; This is file monitor.  This file creates a color display screen on the
;; LISP machine external color monitor.  The display screen contains a
;; 2-D representation of the test pool environment for the AUV dynamic
;; model.  During mission execution, AUV position data is sent to this
;; file from file ap3 and the AUV's track is plotted using a colored icon.

;;DEFINE VARIABLES

(DEFVAR *display-window*)
(DEFVAR *display-window-array*)
(DEFVAR *display-window-width*)
(DEFVAR *display-window-height*)
(DEFVAR *display-window-position*)
(DEFVAR *display-window-screen*)
(DEFVAR *display-window-pos*)

(DEFVAR *main-screen*)
(DEFVAR *screen-alu*)
(DEFVAR *start-alu*)
(DEFVAR *goal-alu*)
(DEFVAR *icon-alu*)
(DEFVAR *grid-alu*)
(DEFVAR *letter-alu*)
(DEFVAR *legend-box-alu*)

(DEFVAR *x-start*)
(DEFVAR *y-start*)
(DEFVAR scale)
(DEFVAR xs)
(DEFVAR ys)
(DEFVAR xg)
(DEFVAR yg)
(DEFVAR xi)
(DEFVAR yi)

;;DEFINE WINDOW AND COLORS

(DEFFLAVOR my-color-flavor()
           (tv:window
            tv:graphics-mixin))

(DEFUN make-color-window
    (window-name position inside-width inside-height
         &rest options &key (superior (color:find-color-screen :create-p t))
         &allow-other-keys)
  (apply #'tv:make-window 'my-color-flavor
         :blinker-p nil
         :borders 2
         :save-bits t
         :expose-p t
         :label nil
         :name window-name
         :position position
```

```
                :inside-width inside-width
                :inside-height inside-height
                :superior superior
                options))

(DEFUN make-display-window ()
 (SETF *display-window*
        (make-color-window "Display-Window"
                            '(50 50) 1150 850))
 (SETF *screen-alu* (SEND color:color-screen
                            :compute-color-alu
                            tv:alu-seta 0.3807 0.5125 1.0))
 (SEND *display-window* :set-erase-aluf *screen-alu*)
 (SEND *display-window* :refresh))

(DEFUN init-display ()
 (clear-scene)
 (draw-box)
 'monitor-display-is-ready)

(DEFUN create-display-window()
 (SETF *main-screen* (SEND *terminal-io* :superior))
 (make-display-window)
 (SETF *display-window-pos*
        (SEND *display-window* :position))
 (SETF *display-window-screen*
        (SEND *display-window* :screen))
 (init-colors)
 'done-init-display-window)

(DEFUN clear-scene ()
 (tv:sheet-force-access (*display-window*)
   (SEND *display-window* :refresh)))

(DEFUN kill ()
 (SEND *display-window* :kill)
 'display-window-killed)

(DEFUN init-colors ()
 (SETF *start-alu* (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 0.406 0.9535 0.2207))

 (SETF *goal-alu*  (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 1.0 0.009008 0.8421))

 (SETF *icon-alu*  (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 1.0 0.0 0.2862))

 (SETF *grid-alu*  (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 0.9054 1.0 0.4847))

 (SETF *letter-alu* (SEND *display-window-screen*
                          :compute-color-alu color:alu-x 0 0 0))
 (SETF *legend-box-alu* (SEND *display-window-screen*
                             :compute-color-alu color:alu-x 0.745 0.7243 0.7976)))
```

```
(DEFUN draw-box ()
 (SEND *display-window*
          :draw-rectangle 1000 500 100 100. *grid-alu*)
 ;;draw vertical lines
 (SEND *display-window*
          :draw-line 225. 100. 225. 600. *icon-alu*)
 (SEND *display-window*
          :draw-line 350. 100. 350. 600. *icon-alu*)
 (SEND *display-window*
          :draw-line 475. 100. 475. 600. *icon-alu*)
  (SEND *display-window*
          :draw-line 600. 100. 600. 600. *icon-alu*)
 (SEND *display-window*
          :draw-line 725. 100. 725. 600. *icon-alu*)
 (SEND *display-window*
          :draw-line 850. 100. 850. 600. *icon-alu*)
 (SEND *display-window*
          :draw-line 975. 100. 975. 600. *icon-alu*)
 ;;draw horizontal lines
 (SEND *display-window*
          :draw-line 100. 225. 1100. 225. *icon-alu*)
 (SEND *display-window*
          :draw-line 100. 350. 1100. 350. *icon-alu*)
 (SEND *display-window*
          :draw-line 100. 475. 1100. 475. *icon-alu*))


;(DEFUN draw-legend-box ()
; (SEND *display-window*
;          :draw-rectangle 400 100 400 650 *legend-box-alu*)
; (SEND *display-window* :draw-filled-in-circle 550 700 20 *start-alu*)
; (SEND *display-window* :draw-filled-in-circle 650 700 20 *goal-alu*)
; (LET ((sx 540)
;          (sy 710)
;          (gx 640)
;          (gy 710))
;    (SEND *display-window* :draw-string "S"
;             sx sy (+ 1 sx) sy t '(:fix :italic :large)
;             *letter-alu*)
;    (SEND *display-window* :draw-string "G"
;             gx gy (+ 1 gx) gy t '(:fix :italic :large)
;             *letter-alu*)))


(DEFUN draw-icon (x y)
 (SEND *display-window* :draw-filled-in-circle x y 10 *icon-alu*))

(DEFUN draw-start-pos (x y)
 (SETF scale 0.694)
 (SETF xs (+ (* x scale) 100))
 (SETF ys (+ (* y scale) 100))
 (SEND *display-window* :draw-filled-in-circle xs ys 20 *start-alu*))
```

```
(DEFUN draw-goal-pos (x y)
 (SETF scale 0.694)
 (SETF xg (+ (* x scale) 100))
 (SETF yg (+ (* y scale) 100))
 (SEND *display-window* :draw-filled-in-circle xg yg 20 *goal-alu*))


(DEFUN move-icon (x y)
 (setf scale 0.694)
 (setf xi (+ (* x scale) 100))
 (setf yi (+ (* y scale) 100))
 (draw-icon xi yi))


;;;main body
;;;prepare monitor

(create-display-window)
(init-display)
```

```
;;; -*- Mode: LISP; Package: USER; Base: 10; Syntax: Common-lisp -*-


; This is the file best.  This lisp code accepts start/goal positions
; from ap3.lisp and uses a best-first search algorithm to calculate
; the appropriate path from start to goal.




(defun get_real_path (start finish autodepth grain)
 (append
   (remove_last (cons start (cdr (best start finish autodepth grain))))
   (list finish)))

(defun remove_last (L)
  (cond
    ((= 1 (length L)) nil)
    (t (cons (car L) (remove_last (cdr L))))))

(defun best (start finish autodepth grain)
 (setq start (list (nearest_20 (car start)) (nearest_20 (cadr start))))
 (setq finish (list (nearest_20 (car finish)) (nearest_20 (cadr finish))))
 (setq autodepth (round_num autodepth))
 (best1 (list (list start)) finish autodepth grain))

(defun best1 (queue finish autodepth grain)
 (cond ((null queue) nil)
          ((equal finish (caar queue))
           (reverse (car queue)))
          (t (best1 (sort (append (expand_node (car queue) autodepth grain)
                                        (cdr queue))
                              #'(lambda (x y) (closerp x y finish)))
                       finish autodepth grain))))

(defun expand_node (path autodepth grain)
 (remove-if
   #'(lambda (path) (member (car path) (cdr path)))
   (mapcar #'(lambda (child) (cons child path))
           (successor (car path) autodepth grain))))

(defun successor (position autodepth grain)
 (setq l nil)
 (cond
    ((< autodepth (get_water_depth (car position) (+ (cadr position) grain)))
     (setq l (cons (list (car position) (+ (cadr position) grain) )  l))))
 (cond
    ((< autodepth (get_water_depth (car position) (- (cadr position) grain)))
     (setq l (cons (list (car position)       (- (cadr position) grain)) l))))
 (cond
    ((< autodepth (get_water_depth (+ (car position) grain) (- (cadr position) grain)))
     (setq l (cons (list (+ (car position) grain) (- (cadr position) grain)) l)) ))
 (cond
     ((< autodepth (get_water_depth (+ (car position) grain) (cadr position)))
      (setq l (cons (list (+ (car position) grain) (cadr position)) l))))
 (cond
     ((< autodepth (get_water_depth (+ (car position) grain) (+ (cadr position) grain)))
      (setq l (cons (list (+ (car position) grain) (+ (cadr position) grain)) l))))
 (cond
     ((< autodepth (get_water_depth (- (car position) grain) (- (cadr position) grain)))
      (setq l (cons (list (- (car position) grain) (- (cadr position) grain)) l))))
 (cond
     ((< autodepth (get_water_depth (- (car position) grain) (cadr position)))
      (setq l (cons (list (- (car position) grain) (cadr position)) l))))
 (cond
     ((< autodepth (get_water_depth (- (car position) grain) (+ (cadr position) grain)))
      (setq l (cons (list (- (car position) grain) (+ (cadr position) grain)) l)))))

(defun closerp (a b with_respect_to)
 (< (get_node_distance (car a) with_respect_to)
   (get_node_distance (car b) with_respect_to)))

(defun get_node_distance (n1 n2)
 (sqrt (+ (square (- (car n1) (car n2)))
            (square (- (cadr n1) (cadr n2))))))

(defun get_water_depth (x y)
 (cond
    ((> 10 (distance x y 205 205)) 0)         ;Sub simulator island
    ((> 20 (distance x y 205 205)) 50)         ;Sub simulator shoals.
    ((> 30 (distance x y 205 205)) 0)         ;Six AUV simulator mines follow.
    ((> 30 (distance x y 200 1202)) 0)
    ((> 30 (distance x y 350 652)) 0)
    ((> 30 (distance x y 380 902)) 0)
    ((> 30 (distance x y 535 227)) 0)
    ((> 30 (distance x y 560 1052)) 0)
    (t (* 0.5 (square (distance x y 205 205)))))    ;This is at least the pool depth.

(defun distance (x y x1 y1)
 (sqrt (+ (square (- x x1))
            (square (- y y1)))))

(defun square (x) (* x x))

(defun round_num (number)
 (car (list (round number))))

(defun nearest_20 (number)
 (* 20 (round_num (/ number 20.0))))
```

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER -*-


; This is file sym-iris-comm1. This file facilitates the interprocess
; communications between LISP machines and IRIS graphics workstations.
; This is an upgrade from file sym-iris-comm.
;
; "Talk" is an object to send and to receive data across a network.
;
; usage : (send talk :init-destination-host 'iris2)  ; get remote host object
;         (send talk :start-iris)                    ; make connection
;         (send talk :put-iris data)                 ; send data
;         (send talk :get-iris)                      ; get data from remote host
;         (send talk :stop-iris)                     ; close communication
;         (send talk :reuse-iris)                    ; open closed communication
;         (send talk :change-iris-ports)             ; switch from iris2 full-duplex
;                                                    ; comms to iris5 semi-duplex


(defvar talk)




;
; library functions to be used by flavor conversation-with-iris.
;



(defun convert-number-to-string (n)
  (princ-to-string n))

(defun convert-string-to-integer (str &optional (radix 10))
  (do ((j 0 (+ j 1))
       (n 0 (+ (* n radix) (digit-char-p (char str j) radix))))
      ((= j (length str)) n)))

(defun find-period-index (str)
  (do ((x 0 (+ x 1)))
      ((equal (char str x) (char "." 0))
       x)))


(defun get-leftside-of-real (str &optional (radix 10))
  (do  ((j 0 (1+ j))
            (n 0 (+ (* n radix) (digit-char-p (char str j) radix))))
      ((or (null (digit-char-p (char str j) radix)) (= j (length str)))  n)))

(defun get-rightside-of-real (str &optional (radix 10))
  (do ((index (1+ (find-period-index str)) (1+ index))
      (factor 0.10 (* factor 0.10))
      (n 0.0 (+ n (* factor (digit-char-p (char str index) radix)))))
     ((= index (length str)) n )))
```

```
(defun convert-string-to-real (str &optional (radix 10))
  (+ (float (get-leftside-of-real str radix)) (get-rightside-of-real str radix)))


(defun num-string4 (num)
; num should be less then and equal to 4 digits.
  (let* ((num-string (princ-to-string num))
              (num-of-leading-zeros (- 4 (length num-string)))
              (leading-zeros
                (make-string num-of-leading-zeros
                                         :initial-element (char "0" 0))))
    (concatenate 'string leading-zeros num-string)))


;
; port number definitions:  Iris2 uses full duplex comms so ports are set up for
;      this default.  Iris5 uses semiduplex comms (the same port for send and
;      receive) and will have both ports set to *remote-port1*.
;


(defvar *remote-port1* 1027)              ; this is the remote send port
(defvar *remote-port2* 1026)              ; this is the remote receive port
(defvar *local-talk-port* 1500)           ; this is the local send port
(defvar *local-listen-port* 1501)         ; this is the local receive port



;
;  conversation-with-iris flavor definition
;
;
;  This definition is not restricted to iris, but it can be
;  used with any host as long as the remote host does not
;  already use ports 1027 or 1026 for its own purposes.
;


(defflavor conversation-with-iris ((talking-port-number      *remote-port1*)
                                   (listening-port-number    *remote-port2*)
                                         (local-talk-port-number   *local-talk-port*)
                                         (local-listen-port-number *local-listen-port*)
                                   (talking-stream)
                                   (listening-stream)
                                         (destination-host-object)
                  )
                      ()
                                   :initable-instance-variables)


(defmethod (:init-destination-host conversation-with-iris)
              (name-of-host)
```

```
 (setf destination-host-object (net:parse-host name-of-host)))


(defmethod (:change-iris-ports conversation-with-iris)
          ()
 (setf talking-port-number    *remote-port1*)      ;sets iris5 semi-duplex comm ports.
 (setf listening-port-number  *remote-port1*))


(defmethod (:start-iris conversation-with-iris)
          ()
 (setf talking-stream
          (tcp:open-tcp-stream destination-host-object
                               talking-port-number
                               local-talk-port-number))
 (setf listening-stream
          (tcp:open-tcp-stream destination-host-object
                               listening-port-number
                               local-listen-port-number))
 (terpri)
 (princ "A conversation with the iris machine has been initiated.")
 (terpri))



(defmethod (:reuse-iris conversation-with-iris)
          ()
 (send self :start-iris))



(defun read-string (stream num-chars)
 (let ((out-string ""))
  (dotimes (i num-chars)
   (setf out-string (string-append out-string (read-char stream))))
  out-string))


(defmethod (:get-iris conversation-with-iris)
          ()
 (let* ((typebuffer   " ")
          (lengthbuffer "    ")
     (buffer       " ")
          (buffer-length 1))
  (setf typebuffer
          (read-string listening-stream 1))
  (setf lengthbuffer
          (read-string listening-stream 4))
  (setf buffer-length
          (convert-string-to-integer lengthbuffer))
  (setf buffer
          (read-string listening-stream buffer-length))


  (cond ((equal typebuffer "I") (convert-string-to-integer buffer))
               ((equal typebuffer "R") (convert-string-to-real    buffer))
               ((equal typebuffer "C") buffer)
               (t nil))))




(defun my-write-string(string stream)
 (let* ((num-chars (length string)))
   (dotimes (i num-chars)
     (write-char (aref string i) stream))))


(defmethod (:put-iris conversation-with-iris)
               (object)

 (let* ((buffer (cond
                         ((equal (type-of object) 'bignum) (convert-number-to-string object))
                    ((equal (type-of object) 'fixnum) (convert-number-to-string object))
                         ((equal (type-of object) 'single-float) (convert-number-to-string object))
                         ((equal (type-of object) 'string) object)
                         (t "error")))

          (buffer-length  (length buffer))

          (typebuffer     (cond ((equal (type-of object) 'bignum) "I")
                                   ((equal (type-of object) 'fixnum) "I")
                                   ((equal (type-of object) 'single-float) "R")
                                   ((equal (type-of object) 'string) "C")
                                   (t "C")))

          (lengthbuffer   (convert-number-to-string buffer-length)))


   (my-write-string typebuffer talking-stream)
   (send talking-stream :force-output)


         (if (= (length lengthbuffer) 4)
            (write-string lengthbuffer talking-stream)
            (write-string (num-string4 lengthbuffer) talking-stream))


         (send talking-stream :force-output)


         (my-write-string buffer talking-stream)
         (send talking-stream :force-output)

         ))
```

```
(defmethod (:stop-iris conversation-with-iris)
             ()
 (send listening-stream :close)
 (send talking-stream :close)
 (terpri)
 (princ "A conversation with the iris machine has been closed.")
 (terpri))


(setf talk (make-instance 'conversation-with-iris))



(defun choose-iris (*host-name*)
 (cond
  ((equal *host-name* 'iris2)
   (setq *host-name* 'iris2)
   (send talk :init-destination-host *host-name*)     ;use iris2 as default output.
   (terpri)
   (princ "Iris2 communications selected.")
   (terpri))
  ((equal *host-name* 'iris5)
   (setq *host-name* 'iris5)
   (send talk :init-destination-host *host-name*)
   (terpri)
   (princ "Iris5 communications selected.")
   (terpri))))

(defun test-iris2()                          ;use these two functions to test
    (equal *host-name* 'iris2))                 ;which iris machine is the output.
                                         ;an example of this is the get_water_
(defun test-iris5()                          ;depth function in best.lisp.
    (equal *host-name* 'iris5))

(defun start-con()
 (send talk :start-iris))

(defun get_data()
 (send talk :get-iris))

(defun send_float(single-float)
 (send talk :put-iris single-float))

(defun send_string(string)
 (send talk :put-iris string))

(defun end-con()
 (send talk :stop-iris))

(defun restart()
 (send talk :reuse-iris))
```

REFERENCES

1. Hawkes, G. and Earle, S., "DEEP FLIGHT: A New Approach to Autonomous Underwater Search and Survey Vehicles," *Unmanned Systems,* v. 5, No. 4, Spring 1987.

2. Bane, G. and Ferguson, J., "The Evolutionary Development of the Military Autonomous Vehicle," *Proceedings of the Fifth International Symposium on Unmanned Submersible Technology*, v. 5, June 1987.

3. Hartman, P., "Practical Applications of Artificial Intelligence in Naval Engineering," *Naval Engineers Journal,* v. 100, No. 6, November 1988.

4. Daniels, J., "Artificial Intelligence: A Brief Tutorial," *Signal,* June 1986.

5. D'Ambrosio, B., "Expert Systems-Myth or Reality?," *BYTE*, pp. 275-282, January 1985.

6. Kumara, S. Soyster, A. and Kashyap, "An Introduction to Artificial Intelligence," *IE*, pp. 9-20, December 1986.

7. Waldrop, M., "The Necessity of Knowledge," *Science*, pp. 1279-1282, March 1984.

8. MacPherson, D., *A Computer Simulation Study of Rule-Based Control of an Autonomous Underwater Vehicle,* Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1988.

9. Nordman, D., *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.

10. Jiang, J. and Doraiswami, R., "Information Acquisition in Expert Control System Design Using Adaptive Filters," *Proceedings of the IEEE International Symposium on Intelligent Control*, 1987.

11. Busby, F. and Vadus, J., *Status and Trends in Autonomous Underwater Vehicles (AUV) Research and Development in the U.S.A.*, National Oceanic and Atmospheric Administration, Rockville, MD, September 1989.

12. Ura, T., "Free Swimming Vehicle PTEROA For Deep Sea Survey," *Intervention '89 Conference and Exposition,* 1989.

13. Durham, J., *Engineering Intelligent Undersea Vehicles*, Naval Ocean Systems Center, San Diego, CA, June 1989.

14. Durham, J., Gillcrist, B., and Heckman, P., *A Testbed Processor for Embedded Multi-Computing*, Naval Ocean Systems Center, San Diego, CA, August 1989.

15. Durham, J., Heckman, P., Bryan, D., and Reich, R., *Eave-West: A Testbed for Plan Execution*, Naval Ocean Systems Center, San Diego, CA, July 1988.

16. Wolfgang, D., Dear, T., and Galbraith, C., *Expert Systems for the Technical Professional*, John Wiley & Sons, Inc., 1987.

17. Keller, R., *Expert Systems Technology: Development and Application*, Yourdon Press, Englewood Cliffs, N. J., 1987.

18. Alexander, T., "The Next Revolution in Computer Programming," *Fortune*, pp. 81-86, October 1984.

19. Shortliffe, E., *Computer Based Medical Consultation: MYCIN*, Elsevier, New York, 1976.

20. Duda, R., Hart, P., Konolige, K., and Reboh, R., "A Computer-Based Consultant for Mineral Exploration," *Technical Report: Final Report, SRI Project 6415, SRI International*, September 1979.

21. Wenstrand, D., Stewart, R., Grabowski, D., and Busher, D., "A Multiple Knowledge Base Approach to AUV Mission Control in Naval Applications," *AUVS 87, Proceedings*, July 1987.

22. Verity, J., "The LISP Race Heats Up," *Datamation*, pp. 55-58, August 1986.

23. Robinson, G., "Mainframe Technology in a Micro," *Design News*, pp. 119-124, January 1986.

24. Serlin, O., "MIPS, DHRYSTONES, and Other Tales," *Datamation*, pp. 112-118, 1986.

25. Newt, J., "A Supercomputer on a Single Chip," *Fortune*, September 1986.

26. Norton, R., "Where the U.S. Stands, Computers, Chips, and Factory Automation," *Fortune*, pp. 28-32, October 1986.

27. "Computer Incorporates 64,000 Processors," *Design News*, p.19, July 1986.

28. Williams, T., "Optics and Neural Nets: Trying to Model the Human Brain," *Computer Design*, pp. 47-62, March 1987.

29. Hopfield, J. and Tank, D., "Computing with Neural Circuits: A Model," *Science*, pp. 625-633, August 1986.

30. Tucker, M., "Coming Next from Japan: The Bionic Computer," *Mini-Micro Systems*, pp. 28-30, July 1986.

31. Victor, J., "Bell Labs Models Parallel Processor on Neural Networks," *Mini-Micro Systems*, pp. 43-51, August 1986.

32. MacDonald, G., *Model-based Compensator Design and Experimental Verification of Control Systems for a Model AUV*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1989.

33. Boncal, R., *A Study of Model Based Maneuvering Controls for Autonomous Underwater Vehicles*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1987.

34. Thayler, G., *Automatic Control Systems*, West Publishing Co., St. Paul/New York/LA/San Francisco, pp. 260-261, 1989.

35. Utkin, V., "Variable Structure Systems With Sliding Modes," *IEEE Transactions on Automatic Control*, v. AC-22, No. 2, April 1977.

36. Bane, G. and Ferguson, J., "The Evolutionary Development of the Military Autonomous Vehicle," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 5, pp. 23-27, June 1987.

37. *KEE Software Development System User's Manual*, version 3.0, pp. 1-1 to 1-4, Intellicorp, Mountain View, CA, March 1986.

38. Friend, J., *Design of a Navigator For a Testbed Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1989.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3	Chief Of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, DC 20350-2000	1
4	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	2
5	Curricular Officer, Code 33 Weapons Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
6	Professor Robert B. McGhee Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	8
7	Professor Sehung Kwak, Code 52Kw Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	2

8	Professor Yuh-Jeng Lee, Code 52Le Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
9	Professor A.J. Healey, Code 69Hy Mechanical Engineering Department Naval Postgraduate School Monterey, CA 93943-5000	1
10	Professor R. Christi, Code 62Cx Electrical and Computer Engineering Department Naval Postgraduate School Monterey, CA 93943-5000	1
11	United States Military Academy Department of Geography & Computer Science ATTN: CPT Mark Fichten West point, NY 10996-1695	1
12	Naval Ocean Systems Center Ocean Engineering Division (Code 94) ATTN: Paul Heckman San Diego, CA 92152-5000	1
13	Naval Coastal System Center Navigation, Guidance, and Control Branch ATTN: G. Dobeck Panama City, FL 32407-5000	1
14	Naval Surface Warfare Center ATTN: Hal Cook, Code u25 White Oak, MD 20910	1
15	HQDA Artificial Intelligence Center ATTN: DACS-DMA, LTC A. Anconetoni The Pentagon, Room 1D659 Washington, D.C. 20310-0200	1
16	RADM G. Curtis, Code PMS-350 Naval Sea Systems Command Washington, DC 20362-5101	1

17 Dr. David Y. Tseng 1
Hughes Research Laboratories
3011 Malibu Canyon Rd.
Malibu, CA 90256

18 Research Administration 1
Code 012
Naval Postgraduate School
Monterey, CA 93943-5000

19 NASA Goddard Space Flight Center 1
ATTN: Russell Werneth
Greenbelt Road
Greenbelt, MD 20771

20 MARINTEK 1
ATTN: Svein Kristiansen
Haakon Haakonsons gt. 34
P.O. Box 4125 Valentinlyst
N-7000 Trondheim, Norway

6-19-588

Thesis
R68555 Rogers
c.1 A study of 3-D visualization and knowledge-based mission planning and control for the NPS Model 2 Autonomous Underwater Vehicle.