NPS52-88-004

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A COMPUTER SIMULATION STUDY OF RULE-BASED CONTROL OF AN AUTONOMOUS UNDERWATER VEHICLE

by

David L. MacPherson

June 1988

Thesis Advisor: Robert B. McGhee

Prepared for:
Naval Surface Weapons Center
Silver Spring, Maryland 20903

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

Kneale T. Marshall
Acting Provost

This thesis is prepared in conjunction with research funded by the Naval Postgraduate School under the cognizance of the Naval Surface Weapons Center, White Oak.

The issuance of this thesis as a technical report is concurred by:

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) NPS52-88-004	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (*If applicable*) Code 52	7a NAME OF MONITORING ORGANIZATION Naval Surface Weapons Center
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Silver Spring, MD 20903
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Naval Postgraduate School 8b OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER OM&N Direct Funding
8c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (*Include Security Classification*)

A COMPUTER SIMULATION STUDY OF RULE-BASED CONTROL OF AN AUTONOMOUS UNDERWATER VEHICLE

12. PERSONAL AUTHOR(S)
MacPherson, David L.

13a. TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 June	

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	SUB-GROUP	Artificial Intelligence; Robotics; Graphics; Autonomous Underwater Vehicle

19 ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Man has an ever-increasing desire for machines to do his work for him. Unmanned vehicles that perform routine or hazardous tasks are receiving a great deal of attention. Vehicles for unmanned submersible applications are becoming more feasible as strides are made in very large scale integration of computer hardware. This work focuses on development of algorithms and ideas for the computer control of military Autonomous Underwater Vehicles (AUVs). Both a Lisp machine and a graphics workstation, communicating via an Ethernet network, were used in this thesis to develop AUV simulator software. The emphasis has been placed on developing a computer graphic simulation of the control panel of an AUV and on a family of programs that define AUV missions. An AUV mission is a complete software plan designed to control an AUV as it executes the steps to achieve some goal or objective. AUV missions are executed by this simulator in a fully autonomous mode once certain mission parameters are supplied by a human user.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Robert B. McGhee	22b TELEPHONE (*Include Area Code*) (408) 646-2095 22c OFFICE SYMBOL Code 52Mz

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

Approved for public release; distribution is unlimited.

A COMPUTER SIMULATION STUDY OF RULE-BASED CONTROL OF AN AUTONOMOUS UNDERWATER VEHICLE

by

David L. MacPherson, Jr.
Lieutenant, United States Navy
B.S., Rensselaer Polytechnic Institute, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1988

ABSTRACT

Man has an ever-increasing desire for machines to do his work for him. Unmanned vehicles that perform routine or hazardous tasks are receiving a great deal of attention. Vehicles for unmanned submersible applications are becoming more feasible as strides are made in very large scale integration of computer hardware. This work focuses on the development of algorithms and ideas for the computer control of military Autonomous Underwater Vehicles (AUVs). Both a Lisp machine and a graphics workstation, communicating via an Ethernet network, were used in this thesis to develop AUV simulator software. The emphasis has been placed on developing a computer graphics simulation of the control panel of an AUV and on a family of programs that define AUV missions. An AUV mission is a complete software plan designed to control an AUV as it executes the steps to achieve some goal or objective. AUV missions are executed by this simulator in a fully autonomous mode once certain mission parameters are supplied by a human user.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND AND BRIEF PROBLEM STATEMENT

Autonomous mobile robots can go where man cannot or does not want to go. Robots perform repetitious, labor-intensive tasks more efficiently and with a higher degree of precision than do human workers. They are suitable for performing tasks in hazardous and contaminated environments. Current and potential robotic applications include agricultural harvesting, factory floor material handling and transfer, combat and combat support systems, and planetary exploration vehicles. Such robots must be able to accept mission statements and constraints, plan their actions, execute these plans, perceive and adapt to their environment, and report successes and failures [1].

The U.S. Navy has a demonstrated need for Autonomous Underwater Vehicles (AUV) to perform a variety of undersea tasks [2]. The purpose of this research is to provide a simulator to be used as a tool for research and development of an AUV at the Naval Postgraduate School. This Autonomous Underwater Vehicle Simulator will be used to develop algorithms for the control of the proposed AUV. Much time and trouble has been saved on similar projects by this rapid prototyping technique. New ideas for proposed AUV missions will be tested first on the AUV Simulator before actually programming the AUV itself.

B. THESIS ORGANIZATION

Chapter II reviews some of the previous research work on unmanned submersible vehicles and other robotic projects. A discussion of the uses of modern computer workstations for dynamic system simulation is included.

The objective of this research work in relation to the Naval Postgraduate School AUV project is discussed in Chapter III. A detailed problem statement is included. The mathematical model for the vehicle dynamics is described here including the autopilot and logical levels of control. The mathematical equations for the vehicle motion are derived and explained. The characteristics of the IRIS graphics workstation and Texas Instruments Explorer Lisp machine workstation are also outlined.

A detailed design and description of the rule-based autopilot is included in Chapter IV. The overall software design is outlined and explained in detail.

Numerous experimental missions were written and tested. Chapter V records and explains the results of all the experiments conducted using the simulation model.

Chapter VI summarizes the work done and explains the contributions this research has made to the field of autonomous vehicle research. A proposal for additional work and extensions is included.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

Research on AUVs began in the early nineteen-sixties [3]. However, severe technological limitations caused interest in autonomous vehicles to dwindle in the late nineteen-sixties. Remotely Operated Vehicles (ROV), limited by range to the operator, and manned vehicles, limited by endurance, provided most of the needed capability required throughout the nineteen-seventies.

Recent advances in processor speed, miniaturization, navigation, and high energy density power supplies have brought the AUV concept back to the forefront of technology. In 1986, the U.S. Navy and DARPA joint AUV Study Committee defined 70 military AUV missions [2]. This work focused on difficult missions that were AUV unique.

B. AIRBORNE ROBOTIC VEHICLES

1. Airborne Remotely Operated Device

The Marine Corps Ground-Air Telerobotic Systems (GATERS) program is developing the Airborne Remotely Operated Device (AROD) [4]. This is a shrouded fan, flying platform linked to the ground operator by a fiber optic cable. The first prototype was flown in December 1986.

The AROD vehicle is remotely operated by the AROD Control Station. This is a portable suitcase size control device linked to the flying AROD platform via a fiber optic cable. The flying platform has a video camera for reconnaissance that provides

over-the-horizon battlefield surveillance. The AROD platform will have an effective operating range of two-to-three kilometers.

The AROD system was designed to be easy for an operator to fly and maneuver with little training. The device is teleoperated, hence it has no on-board intelligence. The AROD is ruggedly constructed for deployment in a harsh battlefield environment.

2. Lockheed Aquila

The Lockheed Aquila remotely piloted vehicle is a tailless mid-wing monoplane with a rear-mounted engine driving a ducted composite pusher propeller [5]. It has a 13 foot wingspan and it is powered by a 26-hp, two-cylinder, air-cooled Herbranson piston engine. This vehicle is launched hydraulically from a standard Army five-ton-truck and is recovered by flight into a vertical ribbon barrier. Aquila is being developed to give the Army the ability to visually monitor the area behind the enemy's front lines [5].

The Aquila system is intended to enhance the accuracy of conventional artillery and also to steer laser-guided bombs and projectiles to over-the-horizon high-value targets. It is designed for low acoustic, infrared, and radar signatures to allow it to perform aerial reconnaissance, artillery adjustment and battle damage assessment three or more miles behind enemy lines [6]. These target acquisition and laser designation capabilities are beyond the normal range of ground observers and can be accomplished without risk to pilots [5].

The Aquila's flight path is controlled from an operator's console in the mobile ground station. It can be flown automatically using preplanned subgoals, or under real-time manual control. Navigation is performed by an on-board computer that communicates with a precision tracking antenna on the ground. An inertial navigation

package permits the Aquila to dead-reckon for up to 23 minutes when it is out of contact with the ground station [6]. Additionally, it has the ability to lock on and automatically track moving ground targets such as a tank, and to focus a laser beam precisely on the intended target. This capability was validated by a 100%-hit-ratio for eight rounds of Copperhead laser-guided missiles fired at ground targets [6]. The U.S. Army will continue to evaluate Aquila using a set of testing milestones. The Aquila will need to evolve to meet these milestones and should continue to prove its usefulness in the battlefield.

3. Development Sciences R4E-40 Skyeye

The Development Sciences R4E-40 Skyeye is a remotely piloted vehicle with a twin tail boom layout. It is powered by a modified 38-hp Kawasaki two-cylinder snowmobile engine with an electric starter [7]. The airframe is 14 feet long, has a 17.5 foot wingspan and is constructed of composite materials. The vehicle is launched by a hydro-pneumatic catapult mounted on a five-ton truck. For recovery, the vehicle lands on a 400 foot runway using its landing skids. The Skyeye is being developed as a versatile "truck" that does not need miniaturization or expensive custom payloads. The maker claims that it can carry a 140-lb payload for eight hours [7].

Skyeye has demonstrated its usefulness by carrying a variety of payload types such as daylight television, low-light-level television, several types of Flir cameras and a 35 mm panoramic camera and meteorological package [7]. In addition, it can carry four 2.75-inch rockets for use against soft ground targets. Skyeye officials believe that it could carry General Dynamics FIM-92 Stinger antiaircraft missiles. Electronic payloads

have also been tested that allow for radio frequency jamming as well as communications relay.

The Skyeye is controlled from a mobile command and control shelter mounted on a 2.5-ton truck. A frequency-modulated radio link provides for a maximum control range of about 80 nautical miles using a mobile eight foot ground antenna [7]. The vehicle can be flown manually, with a simple autopilot, or in a programmed mode that allows up to 256 waypoints. Navigation is by dead reckoning in the autonomous mode.

The Skyeye's small radar cross section, negligible infrared signature, and quiet noise signature make it extremely hard to detect. The first Skyeye RPV flew in 1973. Since then, it has been placed in operational use by the Royal Thai Air Force as well as by the U.S. Army in Central America [7].

4. Tomahawk Cruise Missile

The Tomahawk is a long-range cruise missile with an effective operating range of up to several hundred nautical miles. It is a versatile weapon that can be configured as an antiship weapon or as a land-attack cruise missile. In addition, its launch platforms include submarine, surface ship, ground launcher, and aircraft. The Tomahawk can deliver a 1000-lb conventional warhead in the antiship configuration.

In the antiship configuration, the Tomahawk executes a low cruise transit to the preprogrammed target area using its inertial guidance system. The flight altitude over water has been tested as low as five to ten meters [8]. This extremely low flight altitude minimizes its chance of detection by shipboard radar. When it reaches the designated search area, it then climbs to a search altitude and uses an active radar seeker in the nose

of the missile to find the target ship. The missile executes a search pattern in the area of uncertainty.

As a land attack weapon, the Tomahawk uses an inertial guidance system during its approach over water. Over land, it uses a terrain contour matching optical guidance system to steer a course through a series of preprogrammed waypoints. On July 10, 1981, a Tomahawk launched by a submarine off the coast of California demonstrated pin-point accuracy by hitting a target the size of a small three-story building 300 miles inland in Tonapoh, Nevada [8]. Additionally, it has demonstrated very low altitude, terrain following flights over land with an ability to fly 520 miles per hour 100 feet above the ground.

The Tomahawk's very low radar cross section, low-attitude flight and high subsonic speed tend to deny information to air defense systems [8]. Further development and improvements will continue to make Tomahawk an extremely effective weapon system.

C. LANDBASED ROBOTIC VEHICLES

1. High Mobility Multi-purpose Wheeled Vehicle

The Marine Corps Ground-Air Telerobotic Systems (GATERS) program is developing the high mobility multipurpose wheeled vehicle (HMMWV) configured as a teleoperated vehicle (TOV) [4]. This system consists of a control van and three remote vehicles. Three operators in the control van each control one remote TOV linked to the van by 30 kilometers of fiber optic cable. This system was first tested in March 1987.

The TOV vehicle is a Jeep with an on-board video camera to provide the operator with a look out the windshield via the fiber optic link. The TOV has the on-

board control equipment to allow the remote operator to drive the TOV. The vehicle control system was designated to allow the remote operator to drive the vehicle with very little training.

The TOV was built with four distinct applications in mind: reconnaissance and surveillance, direct fire, control of supporting arms, and NBC (nuclear, biological and chemical) detection.

2. FMC Corporation Autonomous Vehicle

The FMC Corporation Autonomous Vehicle is a modified M113 armored personnel carrier equipped with self-contained power supplies, air conditioning, and computing and communications hardware [1]. Since 1982, this vehicle has been a technology testbed for the development and demonstration of an autonomous vehicle capable of real-time operation both on- and off-road. The system has demonstrated mission planning, route planning with execution at eight km/hr, obstacle detection and avoidance at 15 km/hr, and road following at 24 km/hr [1]. Off-board processing by a transportable command center has allowed developers to focus the program effort on developing system and hardware architecture, algorithms, and software. A five-channel microwave communications system provides the information exchange link between the vehicle and the command center. Four channels send video data from the vehicle to the command center and the fifth channel provides a link for teleoperated vehicle control. Fully autonomous navigation and teleoperation are the two modes of vehicle control.

The FMC vehicle control software consist of five major interconnected subsystems that are called the Planner, Observer, Mapmaker, Pilot and Vehicle Control [9]. These subsystems act in concert to provide fully autonomous operation.

The Planner consists of a mission-planning component that resides on a Symbolics 3600 and two route planners: a symbolic planner that also resides on the Symbolics 3600 and a more conventional pixel path planner installed on a Sun 3 workstation. The Planner uses digitized terrain elevation and feature information to generate a contiguous series of polygonal free space segments from the starting point to the goal. The Planner can also accept mission requirements such as minimizing detection of the vehicle. These requirements are accepted as constraints during the computerized planning of the global path.

The autonomous navigator is comprised of the Observer, Mapmaker, and the Pilot modules that reside on a Sun 3/170 Unix workstation. The Observer uses the segment path generated by the Planner and visual and sonic sensors for obstacle detection to refine the path planned by the Planner. The Mapmaker, in turn, uses this information to generate a Pilot Map. This Pilot Map is very detailed, but very localized, and therefore must be updated often.

The Pilot uses the Pilot Map to guide the vehicle along a dynamically feasible route via the Vehicle Controller Subsystem. This Pilot is designed to respond quickly in an environment with wide passages separating obstacles. The real-time reflexive Pilot analyzes the Pilot Map and generates vehicle control commands in less than 20 milliseconds. In a more restricted environment, such as a forest, a more intelligent but slower copilot takes over and uses a line model algorithm to guide the vehicle [1].

3. Martin Marietta Autonomous Land Vehicle

The Martin Marietta Autonomous Land Vehicle (ALV) platform is an eight-wheeled all-terrain vehicle designed by Standard Manufacturing, Inc. This mobile

platform is to be used as a testbed to integrate and demonstrate strategic computing technologies. This vehicle platform was chosen since it is large enough to house the on-board computers and associated electronics planned through the 1989 demonstrations [10]. This project is aimed at advancing and demonstrating the state of the art in autonomous navigation and tactical decisionmaking. A series of progressively more difficult demonstrations were selected to drive the development of technology in artificial intelligence, image understanding, and advanced computer architecture.

The on-board computer hardware architecture includes an Intel multiprocessor system, for the Reasoning Subsystem and the Pilot, and a VICOM image processor, to support the Perception Subsystem. The Intel multiprocessor system consists of an 80286/80287 master processor, an 80816 navigation processor, an 8086 vehicle control processor and an 8089 multichannel controller [10]. The VICOM image processor is a special-purpose computer system designed to rapidly process digital images. It is used to process both video and range data.

Autonomous mobility in a dynamic, unstructured environment requires that a system sense its environment, model critical features using the sensed data, reason about the model to determine the mobility path, and control the vehicle along that path [10]. To sense its environment, an RCA color video TV camera and a laser range scanner are used. The Perception Subsystem combines the results of video processing and range data processing to provide a 3-D scene model to the Reasoning Subsystem.

The Reasoning Subsystem receives the mission goals and constraints from a human user and coordinates all operations of the Autonomous Land Vehicle. The 3-D scene models from the Perception Subsystem are converted into smooth trajectories that

can be passed to the Pilot to drive the vehicle. The Reasoning Subsystem consists of a goal seeker that selects specific goals for the vehicle to accomplish, a navigator that generates trajectories, and a knowledge base that maintains a map of the test area.

The Pilot takes trajectory commands from the Reasoning Subsystem and converts them into a sequence of steering commands to drive the vehicle. The Pilot operates in one of three modes: drive forward, counterrotate, and stop.

D. REMOTELY OPERATED UNDERWATER VEHICLES

1. Advanced Underwater Search System

The Advanced Underwater Search System (AUSS) is a free-swimming search platform which communicates with its surface support craft via an acoustic telemetry link [11]. This is a research vehicle built by the Naval Ocean Systems Center, San Diego, to fulfill the U.S. Navy's requirement to search for and locate objects on the seafloor at any depth.

The AUSS vehicle is battery-powered and has forward-looking and side-scan sonars as well as a video camera and a 35mm still camera. When operating, the AUSS follows a search pattern pre-programmed in its on-board computer. Adjustments are made by commands sent via the acoustic telemetry system from an operator on the surface. During search transits, surface operators view sonar and TV camera monitors and interrupt the search pattern if an object on the bottom requires closer scrutiny. The vehicle is directed to hover above the object while it is inspected and photographed. The location of the object is recorded and the AUSS returns to its automatic search mode to complete its search pattern.

2. Advanced Tethered Vehicle

The Advance Tethered Vehicle (ATV), also known as EAVE-West is another research vehicle at the Naval Ocean Systems Center, San Diego [11]. This system consists of a tethered open frame platform with thrusters and control devices for motion, and a work package including manipulators, tools, and a TV camera. The tether cable provides the power and an optical fiber data link between the ATV and the surface platform.

The work package provides three manipulators. One for gripping, and the other two for complex work tasks including using tools. The TV camera system transmits close-up pictures of the work site to the remote operator on the surface.

For navigation, the ATV utilizes a deployed field of acoustic transponders. A forward looking sonar and the TV camera are used for the final work site approach.

3. Dolphin

The Dolphin is an unmanned, untethered semi-submersible designed for hydrographic research for the Canadian Hydrographic Service (CHS) [12]. This vehicle was originally developed as a replacement for the 30 foot manned launches used by the CHS for echo sounding off the continental shelf. The Dolphin's greater stability and endurance and its reduced manpower requirement make it a logical choice for replacement of manned launches [12].

The Dolphin is a radio-controlled vehicle remotely controlled from a console aboard the mothership via a 9600 bps telemetry link. It is a snorkeling, diesel powered vehicle capable of speed up to 16 knots and can operate in seas up to sea state six. The Dolphin system is expanding from three to eight vehicles per mothership. Six to eight

vehicles per mothership has been determined to be an optimal ratio for hydrographic surveying by the CHS.

Each vehicle has two on-board computers. A Vehicle I/O Computer (VIO) and a Vehicle Active Controls Computer (VAC) provide the control system for the Dolphin. The VIO computer maintains two-way communication between vehicles and the mothership. The VAC is responsible for maintaining attitude by controlling the vehicle's four control surfaces. Collision avoidance algorithms are being automated to prevent the collision of any two vehicles. Additionally, a silent mode of operation is under development that will allow a vehicle to carry out its mission in the absence of the radio telemetry link.

The U.S. Navy purchased two Dolphin-type vehicles in 1986 for evaluation as possible countermeasure platforms. These vehicles were tested on USN Fleet trials during the summer of 1987.

E. AUTONOMOUS UNDERWATER VEHICLES

1. Knowledge Based System / Experimental Autonomous Vehicle

The Knowledge Based System / Experimental Autonomous Vehicle (KB/EAVE) is an open frame, highly maneuverable, autonomous, unmanned, untethered submersible [13]. This vehicle is also known as EAVE-East. Two vehicles have been built and are undergoing testing at the University of New Hampshire Marine Systems Engineering Laboratory. These vehicles are used primarily as a testbed for new technology.

The KB/EAVE sensor system uses redundancy to check the validity of its sensor inputs. This is critical to an autonomous vehicle operating in a hostile environment.

Depth sonar and a pressure transducer measure depth. A five beam obstacle avoidance sonar enables the system to avoid close-in obstacles when maneuvering. Navigation is performed by the vehicle fixing its position using a long baseline and a short baseline acoustic positioning system. An on-board magnetic compass enables the vehicle to verify its heading.

The computer architecture for the KB/EAVE is divided into a high level and a low level component. The low level portion, comprised of three Motorola 68000 processors, controls motion, processes incoming sensor data, and collects basic sensor data approximately once per second. The high level architecture utilizes five Motorola 68020 series processors to assess the environmental situation, provide supervision and guidance to the low level architecture, and to replan the mission when unexpected events arise. Cycle time for the high level architecture is approximately 100 seconds.

The computer architecture for the KBS/EAVE has required the integration of several hardware and software subsystems [14]. This VME bus based multiprocessor architecture employees several Motorola 68000 series processors. An innovative array of processors is required to overcome problems of execution speed and storage requirements. Software features include PSOS, a real time operating system developed by Software Components Group, and Portable Common Lisp Subset (PCLS) to support the high level symbolic processing. A 800 Mbyte 5 1/4 inch optical disk and a 4 Mbyte RAM board handle the vehicle mass storage requirements.

The high level decision planning functions for cooperating AUVs are to be tested in Lake Winnipesaukee under the NBS-UNH AUV program [15]. Each vehicle is controlled by a hierarchical structure that is very closely related to the division of

responsibility found in a human command hierarchy. A value-driven approach to hierarchical control allows for the decision functions at the upper levels in the hierarchy to be focused almost exclusively on the high level control functions [15]. Hence, the decision responsibility is spread among the levels in the hierarchy to take full advantage of hierarchical control.

The mission-level planner is at the top of the AUV's control hierarchy. Initially, the human user provides the basic mission objectives and priorities to the mission-level planner. The mission-level planner then develops a comprehensive (launch-to-recovery) plan for the group. This plan is optimized by evaluating a number of alternative mission plans and picking the best one. Energy requirements, time urgency, and projected risk factor are used in the optimization. During the mission execution, the mission level planner updates the plan and replans to compensate for unexpected events. Instructions are sent to lower level planners that are concerned only with the next task to be performed [15]. The result is the intelligent adaptive behavior needed for mission execution in a hostile environment.

2. Autonomous Remotely Controlled Submersible

The Autonomous Remotely Controlled Submersible (ARCS) is an unmanned, battery operated, under ice hydrographic survey vehicle developed by International Submarine Engineering (ISE) of Canada. This vehicle was sponsored by the Canadian Hydrographic Service as a cost-effective way to chart waters normally covered by ice [2].

The ARCS vehicle is lowered through a hole in the ice and steers a preprogrammed mapping pattern collecting and storing data on a magnetic tape. The

vehicle's navigational sensors are an acoustic positioning system, a two axis doppler sonar, a depth meter, an acoustic altimeter, and a obstacle avoidance sonar. When the mapping pattern is completed, the ARCS returns to the launch point (a hole in the ice) for retrieval. Obviously, precise navigation and reliability are essential for vehicle retrieval. In typical arctic operation, geopositioning is facilitated using four positioning transponders, two telemetry transducers, and one positioning transducer, all suspended from the ice [2]. Positioning is accurate to five meters.

The on-board computer architecture is designed using a hierarchical command structure similar to that of a naval submarine [2]. Three 16 bit microprocessors are used for on-board control, and an IBM PC at the surface for programming. Data is stored on a 3M sixty megabyte tape drive.

F. USE OF COMPUTER WORKSTATIONS FOR DYNAMIC SYSTEM SIMULATION

The recent emergence of powerful, single user workstations has provided computer scientists with an important research tool. A special purpose workstation, as opposed to a general purpose computer, provides the best environment for the development of complex simulation and control software. Powerful graphics, large storage capacity, innovative software, and networking capabilities have made workstations the computer of choice for dynamic system simulation. Therefore, single user workstations were chosen for the development of the software used in this study.

A graphics workstation is an ideal tool for displaying the dynamic behavior of a computer simulation. Graphics workstations provide specialized hardware and software that enable the user to produce three dimensional, high-resolution color graphics images.

Pipelined, custom graphics hardware allows for real-time animation of graphical objects on the screen. This is a significant aid to understanding the output of a dynamic simulation. Calls to graphics library subroutines and utility commands provide an ideal environment for extensive software development [16].

Artificial Intelligence 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 and Prolog programming languages. Advanced software packages, called expert system shells are available to enhance the user's software development. Networking facilities, large memory capacity, and sophisticated memory management provide the necessary flexibility and speed for the development of intelligent, dynamic simulators [17].

G. SUMMARY

This chapter presents some background information on previous research relevant to this study. A brief survey of airborne and land-based vehicles is included for completeness. Remotely operated underwater vehicles are discussed as a precursor to the development of AUVs. Some of the recent research work on AUVs is covered. The KB/EAVE at the University of New Hampshire's Marine System Engineering Laboratory is the most important and the most relevant topic discussed in this review. The advantages of modern computer workstations for dynamic system simulation are briefly delineated.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

This research is part of a project at the Naval Postgraduate School that will eventually design, build, and test an experimental Autonomous Underwater Vehicle. The purpose of this thesis is to provide a real-time, computer graphics simulation of the AUV. The motivation for an autonomous submarine simulator arises from the need for a research and development tool for the investigation of ideas and algorithms for fully autonomous control of the AUV.

B. VEHICLE DESCRIPTION INCLUDING DYNAMICS

1. The Proposed Vehicle

The proposed vehicle is modeled after the Swimmer Team Delivery Vehicle used by Special Warfare Teams for the covert, submerged delivery of personnel and a payload from a ship to land and back again [18]. A basic idea of the shape of the vehicle and the configuration of the control surfaces was needed to model the vehicle's behavior using a graphics display. However, since the exact vehicle configuration has yet to be determined, the author used operational submarine experience to provide a rough model of basic AUV dynamics. A point-mass approximation is used to simplify the AUV's dynamic motion. In the remainder of this thesis, the term "AUV" will always refer to the simulation of this simplified model.

A "control panel" approach was used to represent the reactions of the AUV simulator to high level commands and its interaction with the environment. The IRIS

color graphics screen was chosen to represent the control panel display [16]. The Texas Instruments Explorer Lisp machine [17] was picked to provide the high level intelligence for AUV's guidance, which includes local obstacle avoidance and overall mission planning.

2. Mathematical Model for AUV Dynamics and Control

The AUV dynamic characteristics used in this study are based on the author's operational submarine experience. No attempt was made to obtain specific dynamic parameters from any actual vehicle. The control is based on the "feel" of the vehicle as it is driven manually. The simulator is meant to handle like a small manned submarine. The simulated vehicle has a single rudder on the stern to control the vehicle's direction, one propeller on the stern for propulsion, sternplanes aft to impart a pitch angle to the hull for large depth changes, and smaller bowplanes forward for fine control of the depth. Bowplanes provide for depth control only and have no effect on the vehicle's pitch angle.

Control of the AUV is available in manual or autopilot mode. The control of the AUV's dynamic motion is based upon its speed and the position of the control surfaces. Constant neutral buoyancy of the simulated vehicle at all depths is assumed throughout this study. The user can manually manipulate the rudder, bowplanes, and sternplanes, and can change the engine speed. To operated unmanned, the AUV has an autopilot control. From the mission requirements, the course, speed, depth, and the location of the next subgoal are determined by the Lisp machine. The autopilot then determines the correct speed and control surface orientation to reach the ordered destination.

The dynamic behavior of the AUV is governed by one acceleration equation, two rate equations, and one attitude equation. The dynamic state of the AUV is determined by updating these equations at approximately 10 Hz.

The value computed for depth rate is a function of the speed, bowplanes angle, and the pitch angle of the hull (which in turn is controlled exclusively by the sternplanes). Specifically, the relationship used is:

$$depth\_rate = speed^2 * (bowplanes\_angle * bowplanes\_depth\_gain) + pitch\_angle * speed \quad (3.1)$$

If the AUV's depth exceeds the bottom depth at a given location, a grounding occurs and the speed goes instantly to zero to simulate an AUV getting stuck in the mud on the ocean bottom. The AUV simulator must be restarted after a grounding.

The AUV's pitch angle is controlled exclusively by the sternplanes. To simplify the simulation, only the steady-state effect of the sternplanes is modelled. Thus, the equation for the AUV's pitch angle is:

$$pitch\_angle = sternplanes\_angle * sternplanes\_pitch\_angle\_gain * speed^2 \quad where\ (depth > 3 feet). \quad (3.2)$$

When the submarine is shallower than three feet, the pitch angle is set to zero to simulate the vehicle leveling off as it approachs the ocean surface.

The rudder is used to control the AUV's course in accordance with the following equation:

$$course\_rate = rudder\_angle * speed * rudder\_angle\_gain \quad (3.3)$$

The AUV's speed control is a model of propeller driven propulsion. To simulate propeller cavitation, the AUV's speed is limited to 8 knots when the vehicle is shallower than 100 feet. At depths greater than 100 feet, the maximum vehicle speed is 12 knots. Vehicle speed is changed 0.1 knot each update cycle until the autopilot speed is reached. This corresponds to an acceleration of one knot/sec/sec.

The AUV's power supply is simulated by a simple battery model. Battery depletion is modelled in accordance with the equation:

$$batterycharge = initial\_batterycharge - (speed^2 * battery\_speed\_gain) \quad (3.4)$$

Maximum vehicle speed is reduced as the battery nears depletion. Table 3.1 summarizes the vehicle's maximum speed as a function of battery depletion.

TABLE 3.1

MAXIMUM AUV SPEED WITH RESPECT TO BATTERY DEPLETION

BATTERY CHARGE(% of full charge)	MAXIMUM AUV SPEED (knots)
100.0 - 12.6	12.0
12.5-9.5	3.0
9.4-7.6	1.5
7.5-0	0.5

The autopilot requires that an autopilot speed, an autopilot depth, and the position of the next subgoal be provided from the guidance level of control. The autopilot manipulates the AUV's control surfaces to reach the next subgoal at the ordered depth and speed. This is the execution level of control for the vehicle.

To manipulate the bowplanes and sternplanes, a depth difference between the present and the desired depth is determined in accordance with the following equation:

$$depth\_difference = autopilot\_depth - present\_depth \tag{3.5}$$

The bowplanes are manipulated in accordance the equations in Table 3.2:

TABLE 3.2

BOWPLANES MANIPULATION WITH RESPECT TO DEPTH_DIFFERENCE

DEPTH_DIFFERENCE (feet)	BOWPLANES ANGLE EQUATION
> 0.1	(depth_difference*0.001) + 12
< -0.1	(depth_difference*0.001) - 12
otherwise	0

Bowplanes are primarily used for fine depth control, but are also used to assist when large depth changes are required.

For depth changes greater than three feet, the sternplanes are used to make the depth changes more rapidly. The following algorithm in Table 3.3 controls the sternplanes motion:

TABLE 3.3

STERNPLANES MANIPULATION WITH RESPECT TO DEPTH_DIFFERENCE

DEPTH_DIFFERENCE (feet)	STERNPLANES ANGLE
> 3	- 10
< 3	+ 10
otherwise	0

To control the simulator's speed, the following equation is used to calculate a speed difference:

$$speed\_difference = autopilot\_speed - present\_speed \tag{3.6}$$

Table 3.4 describes how the vehicle's speed is adjusted based upon speed difference.

TABLE 3.4

AUV SPEED RATE WITH RESPECT TO SPEED_DIFFERENCE

SPEED_DIFFERENCE (Knots)	AUV SPEED RATE
> 0.1	speed_auto_rate_gain
< - 0.1	- speed_auto_rate_gain
otherwise	0

To control the direction of travel of the simulated vehicle, the rudder is used. Autopilot control uses the autopilot course and the present course to determine the required rudder manipulation. The shortest angular distance between the present course and the desire course is calculated to determine the direction of the rudder manipulation. For left rudder, the rudder angle is adjusted to twelve degrees left and for right rudder the

rudder is adjusted to twelve degrees right. If the present and autopilot courses differ by 0.5 degrees or less, then the rudder angle is adjusted to zero.

3. Guidance Level of Control

The guidance level of control is responsible for providing the following control data to the vehicle autopilot (or execution level of control): position of the next subgoal, autopilot speed, autopilot course, autopilot depth, and the mission phase command. These data are the required direction to the autopilot. The guidance level must also process sensor data from the vehicle's sensors and make the appropriate modifications to the next set of control data sent. The Common Lisp code for the guidance level is contained in Appendix B.

4. Mission Level of Control

The mission level of control provides high level guidance and planning for the AUV simulator. The user interface allows the user to select from the available missions and program mission parameters. Mission commands are provided to the guidance level to control overall mission execution. The guidance level must interpret these mission commands and execute the required phase of the mission. The mission level of control is discussed in further detail in Chapter IV.

C. SEA FLOOR SIMULATION

The sea floor model is a submerged cone with its vertex at the coordinates (205, 205). Table 3.5 describes the sea floor model.

TABLE 3.5

SEA FLOOR DEPTH WITH RESPECT TO POSITION

DISTANCE FROM (205, 205) (1 unit = 28 yards)	WATER DEPTH (feet)
< 10.0	0
10.0 - 20.0	50.0
> 20.0	0.2 * ((x - 205) * (x - 205) + (y - 205) * (y - 205))

Uncharted obstacles are simply cylindrical columns that that rise above the ocean bottom to a given height. In other words, an uncharted obstacle is circular portion of the ocean floor raised up closer to the ocean surface. Obstacles are uncharted by virtue of the fact that they appear in the Execution level code (the environment), but not in the Guidance level code (the intelligent perception of the environment). Uncharted obstacles are programmed by modifying the sea floor equations in Table 3.5.

D. WORKSTATION CHARACTERISTICS

1. IRIS Workstation

A Silicon Graphics, Inc. IRIS-2400 Turbo graphic workstation was chosen for the graphics representation of the AUV control panel. The IRIS-2400T is a high-performance, high-resolution 1024x768 color graphics system [16]. It is based on the Unix operating system and has 144 Mbytes of disk storage, 32 bit planes, a hardware matrix multiplier Geometry Pipeline, a floating point accelerator, and a cartridge tape unit. The IRIS hardware consists of three pipelined components: the applications/graphics processor, the Geometry Pipeline, and the raster subsystem. The

graphics capabilities are implemented primarily in hardware, instead of software, as is more commonly done, to enhance the system's efficiency and performance. This feature makes the IRIS particularly well suited for real-time simulation problems.

2. Lisp Machine Workstation

The Texas Instruments Explorer Lisp machine was chosen for the high-level computer control for this study. The Explorer system is an advanced, single-user workstation that provides extensive support for development of large scale, complex programs and for research in new technologies, including artificial intelligence [17]. The programming environment of the Explorer includes high speed symbolic processing, high quality black and white graphics, networking facilities, a large memory capacity, and a sophisticated memory management system. The integrated programming tools afford the user a rich and highly productive programming environment for research and development of complex applications.

E. SUMMARY

This chapter provides a detailed discussion of the problems considered for this study. A description of the proposed vehicle is provided along with the mathematical model used to program the vehicle's dynamic behavior. Additionally, the characteristics of the hardware facilities used in this study are presented.

IV. DESCRIPTION OF THE SIMULATION

A. INTRODUCTION

This chapter presents a description of the simulation software used in this study. The communications software that provides the interface between the IRIS and the Lisp machine is discussed. The execution level of the simulator and its graphics display of the vehicle's control panel are described in detail. The control panel is the means used to tell the user what the AUV is doing to execute its mission. The guidance level of the autopilot is also described. This portion of the autopilot interprets overall mission commands from the supervisor level and directs the execution level. A discussion of the mission level tells how the basic objectives for the mission are obtained from the human user. This input data allows the mission level to develop a specific plan for the intended mission from launch to recovery. A diagram of the hierarchical AUV system architecture is shown in Figure 4.1.

B. COMMUNICATIONS SOFTWARE

The communications software package used in this study is described in [19]. Specifically the Inter-Computer Communications Package, presented in the Appendix to [19], provides standard routines to communicate between different computers connected via Ethernet and Transmission Control Protocol/Internet Protocol (TCP/IP) [20].

The server part of this communications package runs on the IRIS workstation and is written in C. Two ports are used for establishing communication. The ports are

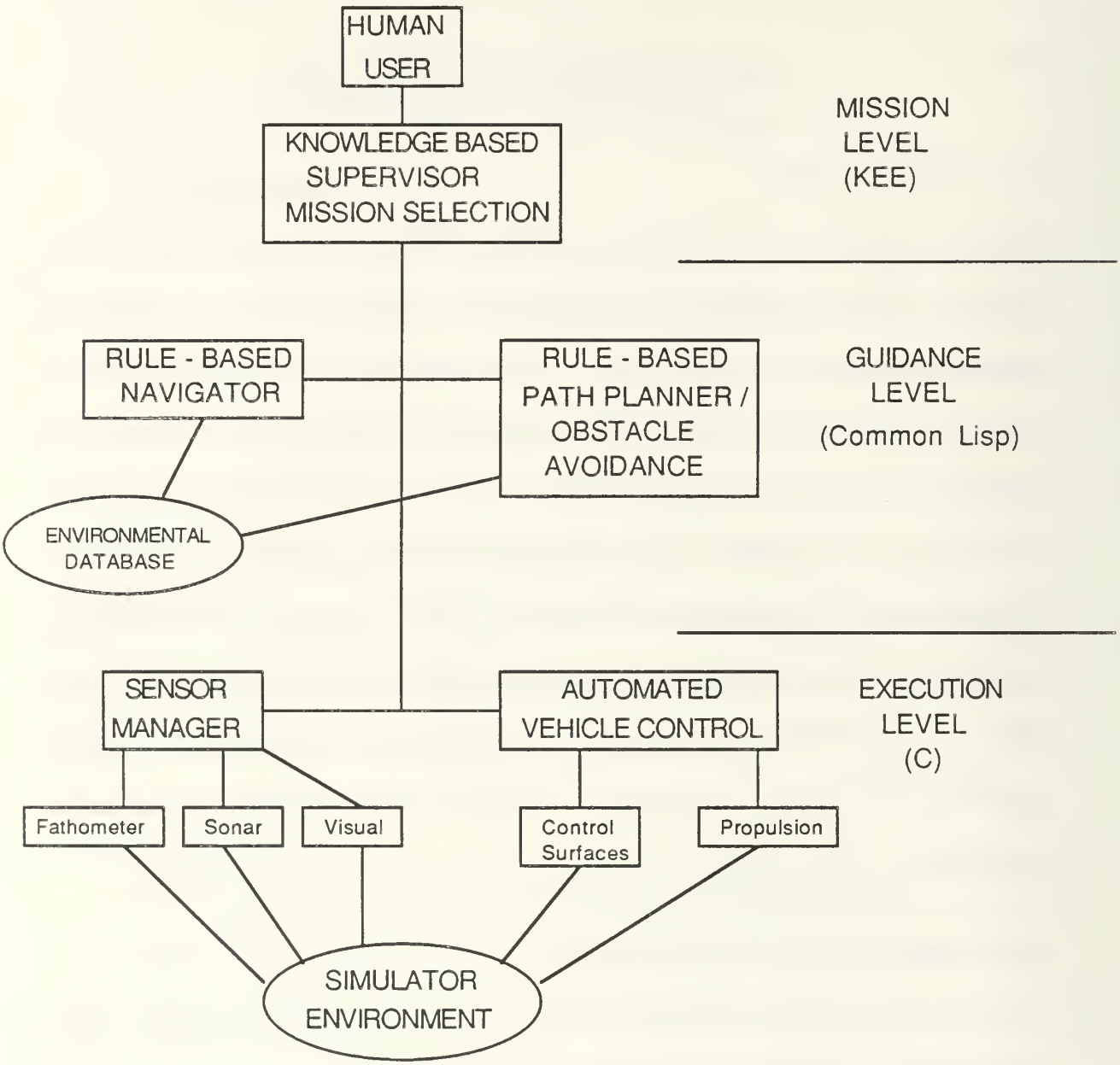


Figure 4.1 HIERARCHICAL SYSTEM ARCHITECTURE

connected to sockets in TCP/IP and a process is spawned to service each socket. A shared memory segment of 1 Mbytes is used to communicate between these two processes and the main loop, look.c, of the simulation program.

The client part of the simulator runs on the TI Explorer Lisp machine and is written in Common Lisp. This portion uses an instantiation of the IP::TCP-HANDLER flavor in the TCP/IP software package [16]. Routines are provided for establishing a connection with a specific computer as well as sending and receiving floating point numbers, integers, and characters.

The information exchanged between the IRIS and the Lisp machine allows the Lisp machine to send commands to control the IRIS and also allows the IRIS to send sensor data back to the Lisp machine for analysis. Information exchange occurs about every three seconds. The IRIS sends the AUV's present course, speed, vehicle depth, depth under keel, and the bearing and range to all sonar contacts every cycle. The Lisp machine then sends the mission phase command, the autopilot course, speed, and depth required to reach the next subgoal, and the position of the next subgoal back to the IRIS. Appendix B contains the program iris-flavor4.lisp#7. This is the software for the Lisp machine side of the communications interface.

C. GRAPHICS DISPLAY OF THE AUV CONTROL PANEL

A diagram of the AUV control panel is shown in Figure 4.2. This is a simplified representation of the instrument control panel on a full-size U.S. Navy Submarine. This display is the primary means for a user to observe the performance of the AUV as it executes its mission.

Figure 4.2 AUV CONTROL PANEL

The upper left corner of the panel is the view out of the periscope. The periscope is automatically raised whenever the AUV's depth is shallower than 50 feet. The periscope allows selection of high or low power optical magnification, and the azimuth and the elevation of the view out of the periscope. Several ships, an island, a target, and two buoys are included in the AUV's environment to provide obstacles for the simulation.

The upper center portion of the control panel is an indicator panel. A battery charge indicator simulates the state of the simulator's power supply. An alarm is provided for a low battery condition. Depth, course, and speed indicators allow the user to follow the AUV's behavior as it reacts to guidance level commands.

The upper right portion of the control panel is the active sonar display. Sonar contacts are displayed as black dots on the sonar screen. The black cursor indicates the AUV's true heading and the green cursor indicates the azimuth to which the periscope is trained.

The lower left portion of the control panel is a chart that displays the AUV's position in its environment. A small red X is used to indicate the AUV's position on the chart. The mission starting point and the goal are labeled START and GOAL respectively when a mission is run in the autopilot mode. The island and the shallow water surrounding it appear near the center of the chart.

The lower center portion of the control panel contains the AUV's hull pitch angle indicator. To the right of this is a warning panel with alarms for shallow water under the AUV's keel and for a grounding of the AUV. Below the warning panel is the AUV's fathometer that indicates the depth of water under the AUV's keel.

The lower right portion of the AUV control panel contains the control surface indicators. The simulated position of the AUV's rudder, bowplanes, and sternplanes is indicated on these instruments.

D. THE EXECUTION LEVEL

The execution level of the AUV is written in C and runs on the IRIS-2400T graphics workstation. This level is the lowest level of vehicle control. Manual keyboard commands are interpreted and executed in the manual operation mode.

In autopilot, the guidance commands control the following: position of the subgoal, autopilot speed, autopilot depth, and the mission phase. These commands are received from the guidance level during each information exchange cycle. The position of each control surface needed to attain the ordered depth, speed, and course is next determined. Finally, the control surfaces are positioned as required. The execution level updates the graphics display approximately ten times per second to reflect the current state of the AUV.

Sensor data are provided to the guidance level for processing and interpretation during the data exchange cycle. The guidance level is responsible for interpreting these data and for making any required modifications to the next set of guidance commands. One possible modification is a shallower autopilot depth, based on fathometer information, to prevent the AUV from grounding in uncharted shallow water. This particular feature is discussed in further detail in Chapter V.

E. THE GUIDANCE LEVEL

The guidance level of the simulation is written in Common Lisp and runs on the Texas Instruments Explorer Lisp machine. This level of control receives overall mission orders from the mission level and provides guidance commands to the execution level. The position of the next subgoal, the autopilot speed, the autopilot depth, and the mission phase command required to execute the present phase of the mission are provided to the execution level during frequent data exchanges via the communications interface. Sensor data, such as sonar contacts, the vehicle's position, and the depth of the water under the keel are received from the execution level during each data exchange. The guidance level processes the sensor data and modifies the guidance command as necessary for the next data exchange.

The guidance level of control is actually a set of modular procedures. Each procedure is designed to accomplish some specific phase of the mission. For instance, the transit procedure provides a best-first search routine [21] for global path planning. A search is required to plot an intended route to a desired subgoal in order to avoid charted obstacles. The guidance level must be given a mission command from the mission level in order to know what procedures to invoke and to tell when they will be needed.

F. THE MISSION LEVEL

The mission level of the AUV is written in KEE and runs on the TI Explorer Lisp machine. This level of control allows the user to select one of several missions. A mission is selected by using the left mouse button to chose one of the leaves of the mission selection tree shown in Figure 4.3. The mission level then takes over and asks the user to enter critical mission parameters for the particular mission chosen. The

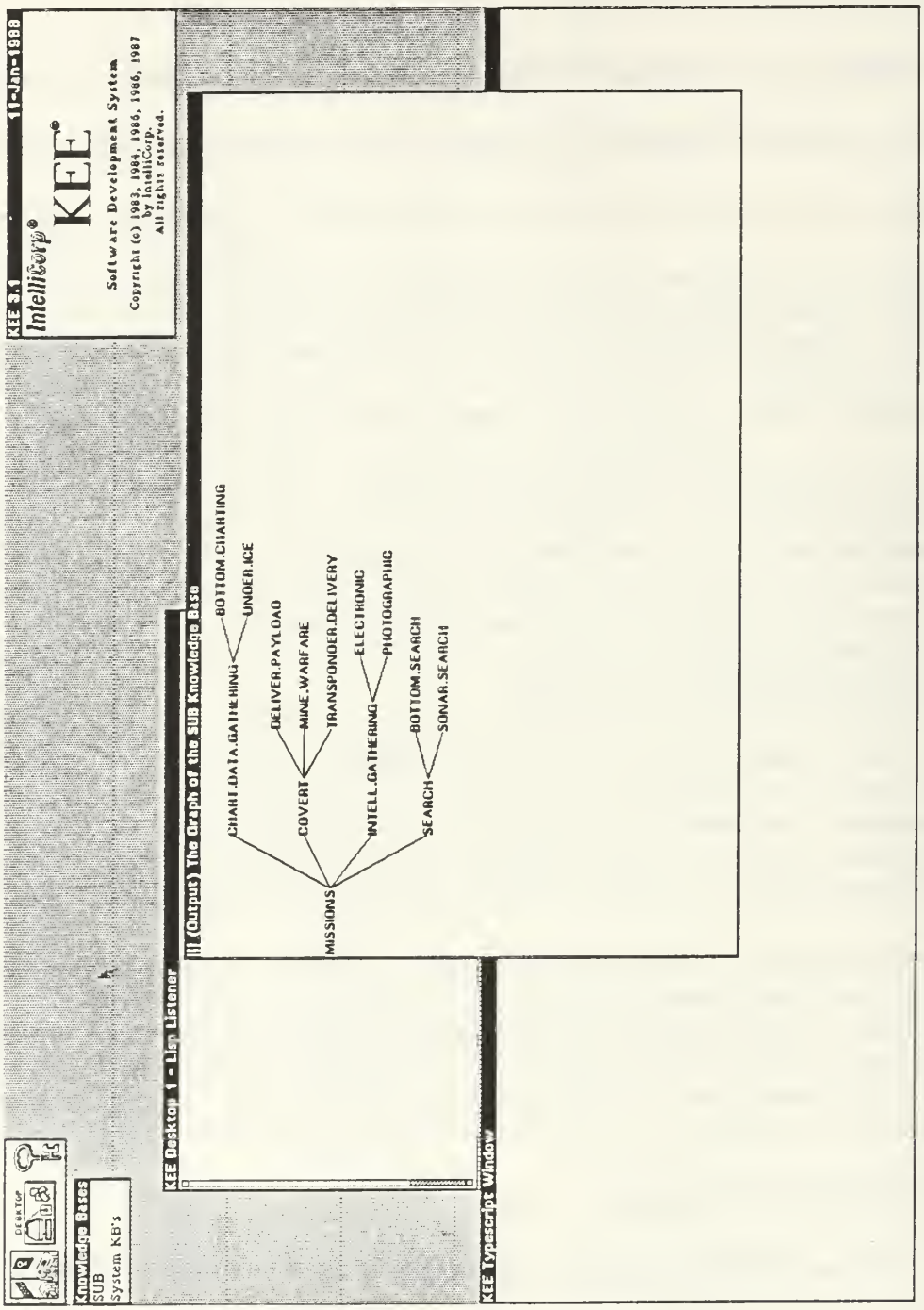


Figure 4.3 KEE MISSION SELECTION TREE

mission level plans the mission specified and then provides the necessary mission commands to the guidance level. Execution of the mission begins automatically once the initial path planning is completed.

G. MISSION TEMPLATE

The mission template concept is an aid to human understanding of the format of the intended AUV mission. The first step in programming a new type of AUV mission is to write a new mission template for the intended mission. The mission template concept is also a means for a user to quickly develop an understanding of the format of a given generic mission. This feature is designed to allow research and development teams to create and modify various AUV missions. A blank mission template is shown in Figure 4.4. To create a proposed generic mission, a programmer first fills out the blanks in the mission template form. This can then be used as a specification to write the program for the intended mission. Of course, once the program is written, the template is no longer needed since the Lisp machine will prompt the user for every entry in the template for the selected mission. Appendix A contains all mission templates created in this study along with a pictorial explanation of each mission scenario. The step numbers in the mission format portion of the templates are the same as the numbers on the diagrams for each mission.

H. USER'S MANUAL

The AUV Simulator is relatively easy to use, providing the user has a basic understanding of the TI Explorer, the KEE expert system shell, and the IRIS 2400T graphics workstation. The following guide assumes this basic understanding.

AUV MISSION TEMPLATE:

NAME:

PURPOSE:

DURATION:

TRANSIT DEPTH:

TRANSIT SPEED:

ON STATION TIME:

ACTION ON STATION:

MISSION FORMAT:

Figure 4.4 AUV MISSION TEMPLATE

To start the IRIS 2400T graphics workstation, "log on" on the side terminal first and then transfer to the directory */work/macpher/subsim.* Type *sub* and then a carriage return to start the simulator. Then press any mouse button after reading the initial display. The AUV simulator is now in the manual control mode at depth, *0 feet;* speed, *0 knots;* and course, *North.*

Manual control of the AUV simulator is effected using the IRIS keyboard and the mouse buttons. To obtain help with manual control, the I-key toggles a *Help Display* on the chart portion of the IRIS graphics screen. This display tells which keys on the keyboard operate the various controls of the simulator.

To steer the vehicle, the left and right arrow-keys manipulate the rudder. The left arrow-key gives a left rudder and the right arrow-key gives a right rudder. The sternplanes are used to control the vehicle's pitch angle; the D-key places dive on the sternplanes and the S-key gives a rise on the sternplanes. The bowplanes are controlled by the B-key for dive and the C-key for rise. The speed is controlled by the Up arrow-key to raise the speed and the Down arrow-key to lower the speed. The date/time display is toggled on and off using the Z-key.

The periscope has several different control features. The periscope magnification is changed using the H-key and the L-key for high and low power respectively. The left mouse button rotates the periscope direction counter-clockwise and the right mouse button rotates it clockwise. The middle mouse button is used to increase the periscope view angle and both the right and left mouse buttons pushed simultaneously will lower it.

To start the autopilot, press the A-key on the graphics keyboard. The side terminal will now indicate that the IRIS is waiting to connect to nps-ccc (this is the TI Explorer Lisp machine).

On the TI Explorer, make sure that the KEE expert system shell software is loaded. If it is not, a cold reboot of the system is necessary to load KEE. An inexperienced user should get help with this step. Once KEE is loaded, "log on" and then press the SYSTEM-key and the K-key simultaneously to go to the KEE Desktop. Next use the mouse to point at the KEE icon (an "old-style" latch-key) in the upper left corner of the screen. Push the left mouse button and a pop-up menu will appear labeled *KEE Commands*. Select *Load KB* by pointing with the mouse to the *Load KB* selection and then pushing the left mouse button. A KEE typescript window will appear asking for the name of the knowledge base to load. Type *aries: macpher; sub.u* followed by a carriage return to load the AUV simulator knowledge base. Loading will take about one minute.

Next use the mouse to point to the word *SUB* in the knowledge base window and press the left button. A pop-up menu will appear labeled *KB Commands*. Again use the left mouse button to select *Display* and roll off of the menu to the right. Another smaller pop-up menu will now appear. Point to the word *Graph* and push the left mouse button. The AUV Simulator Mission Selection Tree will appear in the window labeled *The Graph of the SUB Knowledge Base*.

Load the Common Lisp files required for the guidance level of control by typing *(load "aries: macpher; ap3.lisp")*, *(load "aries: macpher; best.lisp")*, and *(load "aries: macpher; iris-flavor4.lisp")* in the Lisp Listener window. Once these files are loaded, a generic mission may be selected from the Mission Selection Tree.

To choose a mission, use the mouse to point to the desired generic mission. This is one of the leaves of the Mission Selection Tree. Then press the left mouse button to select the generic mission. A menu labeled *Unit Commands* will pop-up. Use the mouse to point to *Send Message* and then press the left mouse button again. A menu labeled *Message Types* will pop-up. Use the mouse to point at the message that starts the desired generic mission type and then press the left mouse button.

The AUV simulator will now ask the user questions about parameters for the specific mission. These questions will appear in the KEE Typescript Window. Type answers to each of these questions followed by a carriage return. When the words *connection with the iris established* appear in the KEE Typescript Window, press a carriage return on the IRIS side terminal. Press carriage return on the IRIS side terminal again after the words *Autopilot course on the first leg is:* appear in the KEE Typescript Window indicating that the initial path planning search is complete. The AUV simulator will now run automatically to execute the programmed mission.

To stop the autopilot, press the Q-key on the IRIS graphics keyboard. The simulator is now back in the manual control mode. The autopilot cannot be restarted at this point due to shared memory constraints. Push all three mouse buttons simultaneously to stop the graphics portion of the simulator.

To restart the TI Explorer side of the AUV simulator, it is necessary to type *(end-con)* in the Lisp Listener Window to break the socket connection between the IRIS-2400T and the TI Explorer. Then it is necessary to type *(restart)* to reestablish the ability to make this connection for the next run of the AUV Simulator.

Prior to restarting the IRIS-2400T after a simulator run, it is necessary to ensure that the socket connection has been broken using the command; *ps ax*. Use the *kill* command to remove any leftover socket demons from the previous trial.

I. SUMMARY

This chapter presents a description of the simulation used in this study. The first section describes the communication software used to facilitate the required communication interface between computers. Next, the graphics display is described in detail using a diagram of the AUV's control panel. Finally, each of the levels of hierarchical vehicle control is discussed. The interrelationships among these software components are stressed. A mission template concept is introduced as an aid to the human understanding of AUV missions. A User's Guide is provided to assist those interested in testing or improving the AUV Simulator software.

V. EXPERIMENTAL RESULTS

A. INTRODUCTION

This chapter provides an evaluation of the experimental results obtained with the AUV software developed in this study. Many simulation runs were carried out to test the AUV's ability to perform the specified missions. The AUV performed all programmed missions successfully. More importantly, the AUV's modular software demonstrated its extensibility as the family of AUV missions was expanded during their development.

B. RESULTS OF VEHICLE SIMULATION

The AUV software provides both manual and autopilot control of the simulator. The simulator is reasonably hard for one operator to control in the manual mode since there are eight keyboard keys required to fully control the simulator's motion. The means of manual control is certainly not intuitive, especially for an operator with no submarine experience.

The autopilot feature is essential for an AUV to operate unmanned. The guidance level of control of the autopilot provides the AUV with a means for moving to the next subgoal at a specified depth and speed. Obstacle avoidance capabilities of the guidance level enhance an AUV's ability to survive in a hostile environment as well as to complete missions despite uncharted obstacles.

The mission level of control of the autopilot provides a means for a human operator to select the desired AUV mission and choose critical mission parameters. The mission template is used as a specification for the programmed missions. Once a mission is

specified and the critical mission parameters are entered, the autopilot then plans that mission and supervises its overall execution.

The AUV successfully demonstrated an ability to plan a path around charted obstacles, to avoid uncharted local obstacles, and to avoid close surface contacts when operating at a shallow depth. These capabilities are critical to the survival of an AUV in a hostile operating area.

An example of the AUV simulator's ability to plan a path around a charted obstacle is shown in Figure 5.1. The TI Explorer has a bottom contour database of the simulated operating area in the file Best.Lisp#4 (see Appendix B). The path planning is accomplished using a modified best-first search [21] of this bottom contour database before the vehicle begins its mission. The path generated is used both for the vehicle going to the goal or mission execution area, and for the vehicle returning from the mission area to the starting point for subsequent recovery.

Figure 5.1 depicts an actual experimental search path from the coordinates (100,100) to (240,240). The path chosen is represented as a list in Common Lisp. The resultant path is represented as ((100,100),(180,240),(200,240),(220,240),(240,240)). This path keeps the vehicle at least 100 feet above the ocean floor throughout the entire transit.

The AUV simulator's ability to avoid uncharted local obstacles is shown in Figures 5.2 and 5.3. Figure 5.2 is a profile plot of the AUV's depth as it encounters an uncharted shallow area of the ocean floor. This area is considered uncharted because it is not represented in the Lisp machine's bottom contour database, although it is represented in the IRIS graphics workstation's bottom contour database.

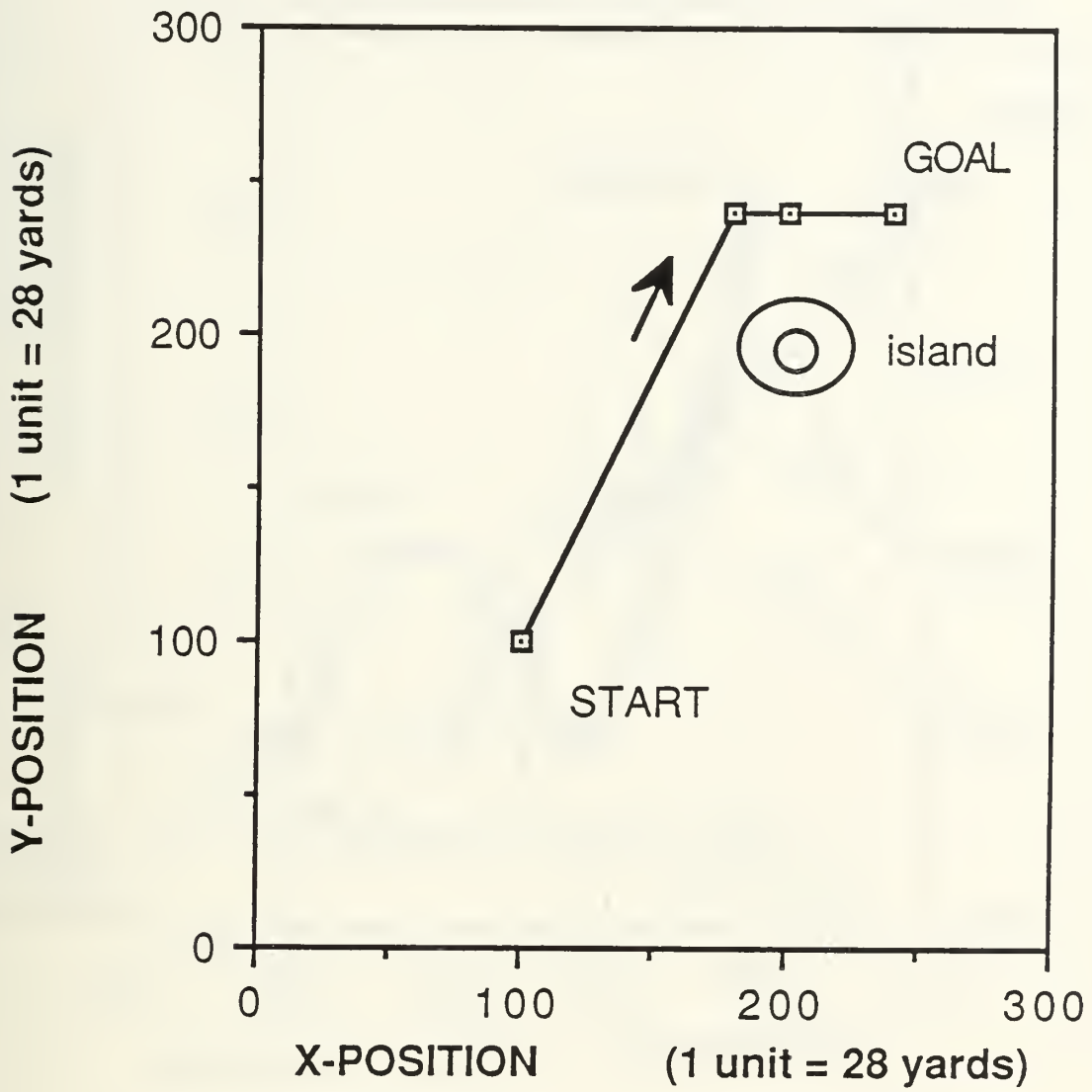


Figure 5.1 OBSTACLE AVOIDANCE USING BEST FIRST SEARCH

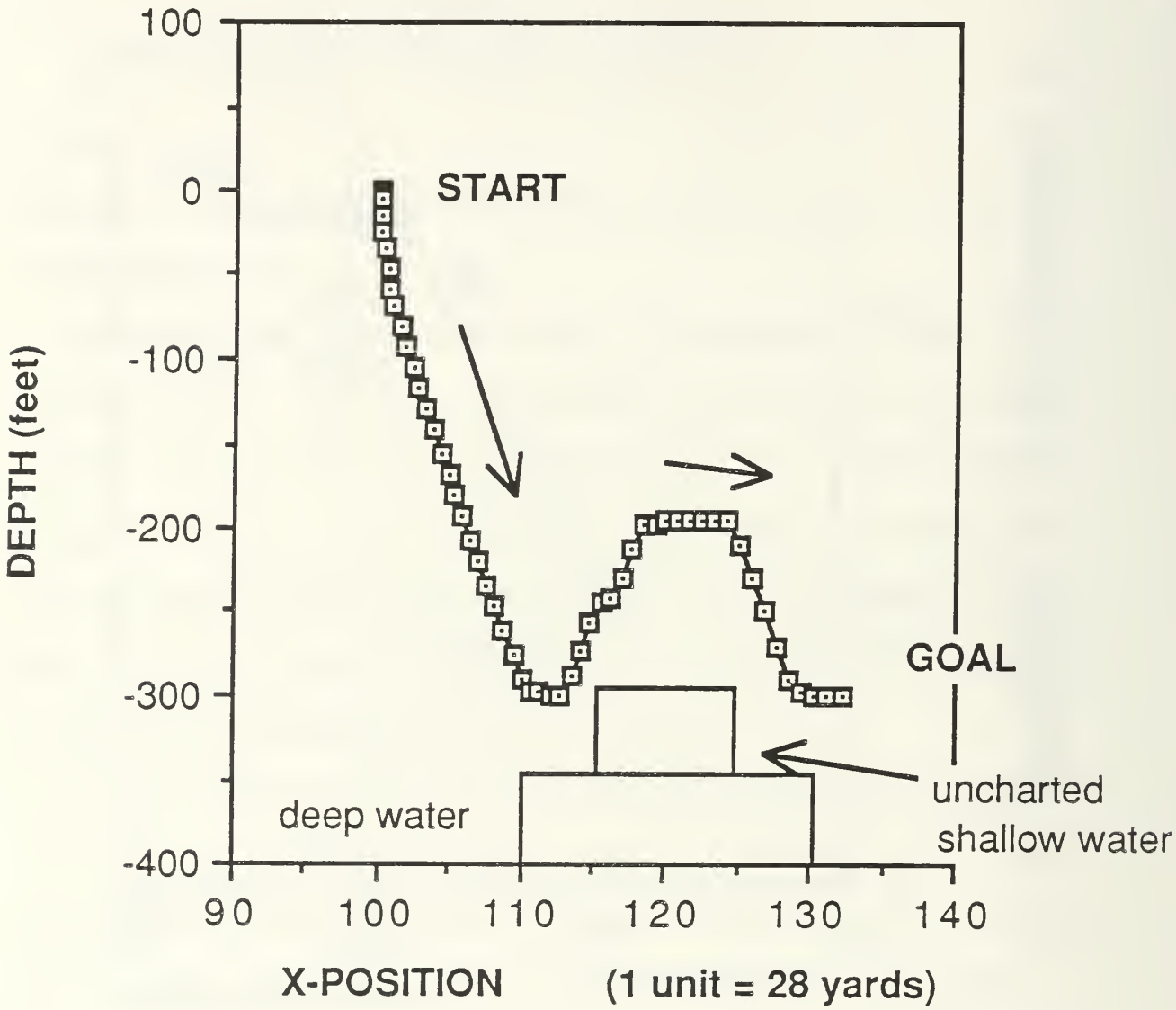


Figure 5.2 PROFILE OF UNCHARTED OBSTACLE AVOIDANCE

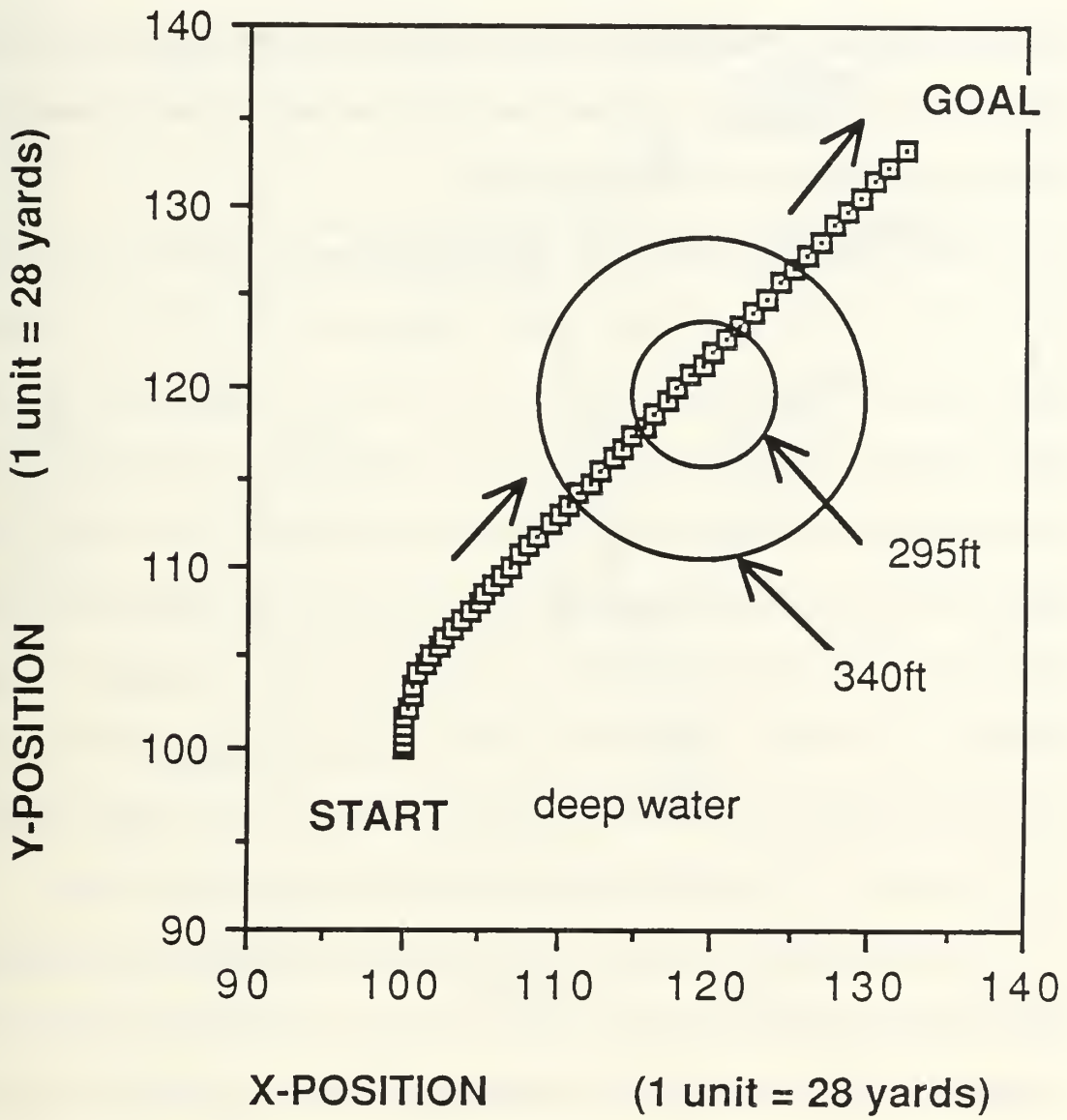


Figure 5.3 TRACK OF UNCHARTED OBSTACLE AVOIDANCE

The AUV simulator's guidance level uses Fathometer information to attempt to keep the AUV at least 100 feet above the ocean floor at all times. This obviously works quite well for a shoaling ocean bottom, but not so well when a cliff is encountered with no bottom shoaling beforehand. A forward scan active sonar model would be needed for the simulator to avoid an uncharted underwater cliff.

A more detailed description of the behavior illustrated by Figures 5.2 and 5.3 is as follows. First, the AUV is ordered to transit from the point (100,100) to (140,140) at a transit depth of 300 feet. Figure 5.2 shows the AUV dive to ordered transit depth. Once this is attained, the AUV encounters water only 340 feet deep. The guidance level of control recognizes that there is only 40 feet of water beneath the keel. A new autopilot depth of 240 feet then is ordered to keep the AUV 100 feet above the ocean floor. Subsequently, water only 295 feet deep is encountered which the guidance level again recognizes as too shallow, and orders the AUV to an autopilot depth of 195 feet. Once the AUV has transited past the shallow water, the guidance level of control orders the AUV back down to 300 feet, the normal transit depth for this particular mission.

The AUV simulator's ability to avoid close surface contacts while operating at shallow depth is depicted in Figures 5.4 and 5.5. Figure 5.4 is a profile plot of the AUV's depth as it encounters a nearby surface contact while operating at periscope depth. The AUV simulator detects the close surface contact on active sonar. Then the AUV executes a dive to 68 feet to avoid collision with the surface contact. Once danger of collision is over, the AUV comes back up to periscope depth. Active sonar operates continuously to detect any new close aboard collision threats during the ascent to periscope depth as well as after the transit at periscope depth continues. Figure 5.5

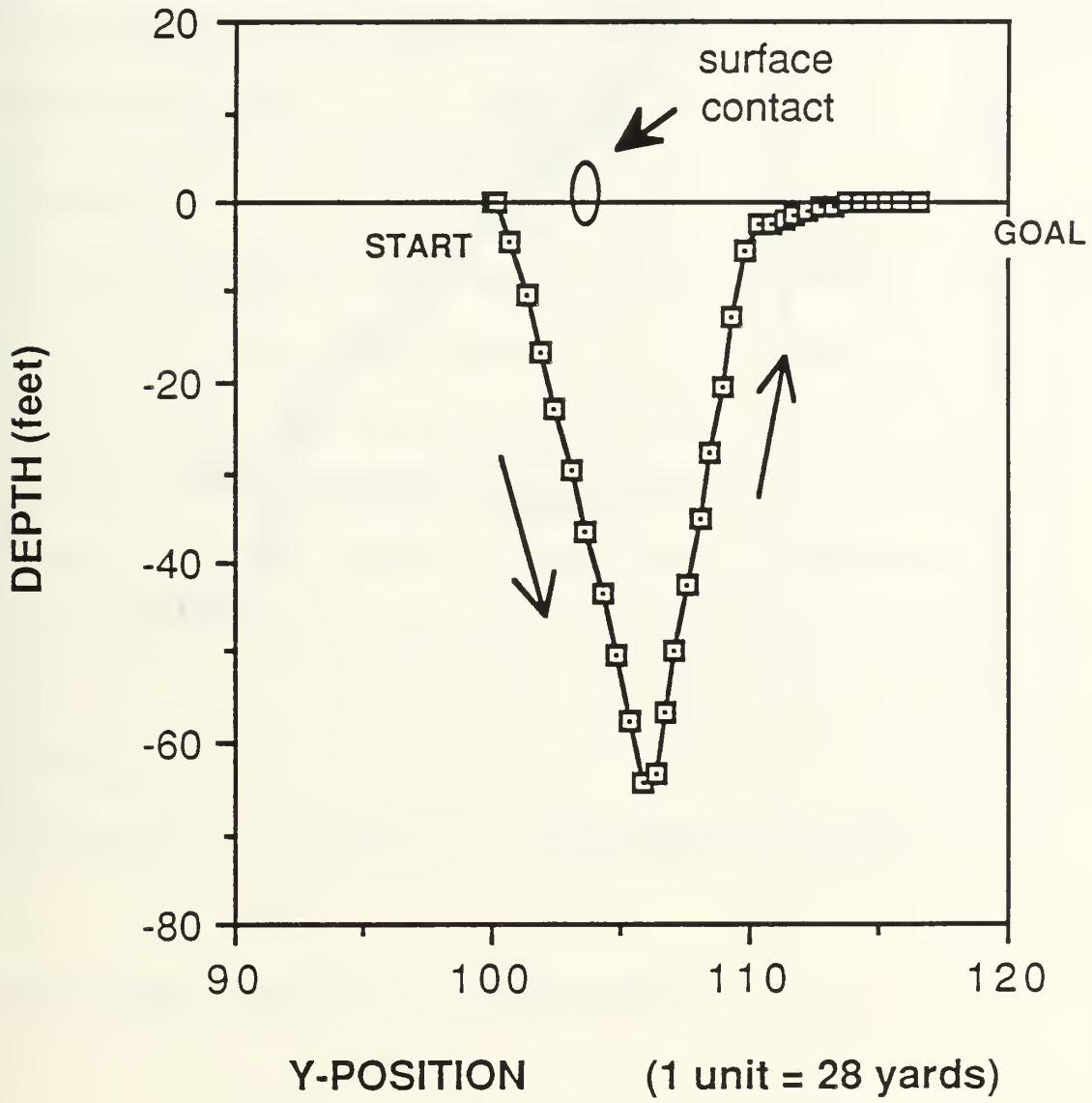


Figure 5.4 PROFILE OF SURFACE CONTACT AVOIDANCE

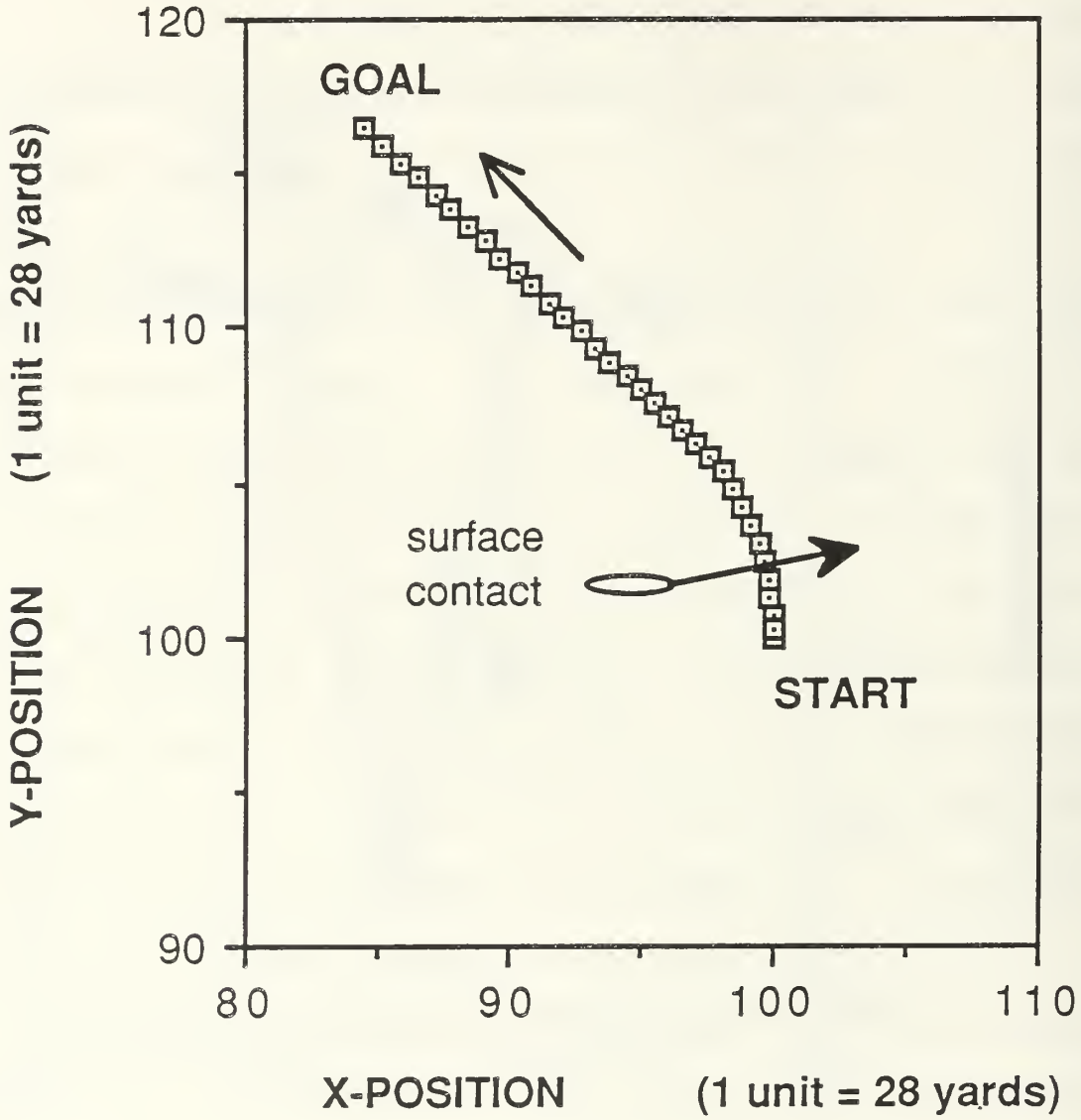


Figure 5.5 TRACK OF SURFACE CONTACT AVOIDANCE

shows the AUV's track from starting position at (100,100) to the goal (80,120). The AUV's maneuver to avoid the surface contact is strictly a depth maneuver. The curve in the track is simply the AUV maneuvering to attain autopilot course from its initial orientation of course North.

C. SUMMARY

This chapter presents an overall evaluation of the AUV simulator experimental results. A brief discussion of AUV control in manual mode and autopilot mode is included. The successful demonstration of the AUV's ability to execute several missions in the mission program family indicates that this simulator is a valuable tool for AUV development. The author's experiences in rapidly specifying and programming new missions suggests that the AUV will become an important software development device for the AUV to be built at the Naval Postgraduate School. The AUV simulator has demonstrated a capability to plan a path around charted obstacles, to avoid uncharted local obstacles, and to avoid close surface contacts while operating at periscope depth.

VI. SUMMARY AND CONCLUSIONS

A. RESEARCH CONTRIBUTIONS

The AUV simulator developed in this study provides an important tool to test new ideas and algorithms for autonomous control of an AUV. Countless hours of development time can be saved in the future by testing and debugging new algorithms before installing them in the real AUV computer.

The introduction of the mission template concept for human mission planning and understanding of AUV missions is an important contribution of this study. The mission template can be quickly written to provide a specification for a generic mission. Once a mission template is written and approved, it provides the programmer with an outline to be used to write the actual generic mission program. Of course, after the program is written, actual specific missions are programmed by first selecting one of the generic missions and then entering mission specific data such as the coordinates for the location of a sonar search.

This study produced a family of programs that is modular in design and easily extensible to allow the creation of future, yet unspecified, AUV missions. Modules are used as building blocks to move the AUV to a desired location, perform critical mission components on station, and then return the vehicle to the starting point for subsequent recovery.

B. RESEARCH EXTENSIONS

Extensions to improve and enhance the capabilities of the AUV simulator of this thesis could be the subject of future research. Periscope vision, using the IRIS pixel recognition feature [16], could be used to improve the photographic reconnaissance mission by allowing the AUV to visually recognize ships, land, etc. This feature would allow the simulator to perform more sophisticated missions such as ship following and identification.

A faster search algorithm for path planning could be implemented. A-star search [21] or a pixel path planner search routine written in C would provide additional speed.

More efficient use of Ethernet data packets on the network interface would improve the simulator's responsiveness to changing environmental conditions. Presently, only four bytes of the minimum length 512 bytes Ethernet packets are being used to transfer data via the communications interface. All data for one machine-to-machine information exchange could be sent via one Ethernet data packet to improve the efficiency across the communications interface.

Better local mission replanning will make the AUV more survivable in a hostile environment. This is perhaps the most difficult area to improve.

More sophisticated sonar modeling could be investigated including side scan and forward scan sonars to enhance navigation and obstacle avoidance and to allow the AUV to locate objects on the ocean floor.

Presently, the AUV does not represent any particular vehicle with respect to its control characteristics. The use of the dynamic characteristics from an existing small

AUV would make the AUV's behavior more realistic and would make the AUV's position and depth control more meaningful. Perhaps a menu driven option to allow the user to select vehicle control characteristics before the simulation is started would be useful.

The AUV simulator's accuracy and resistance to outside influences could be improved by using the actual time increment between state updates in the state update equations. Presently, no measure is made of this time increment. It is just assumed that the simulation will run at about ten updates per second.

A problem was encountered in the AUV's maneuverability when adjusting the vehicle's rudder_angle_gain. To make the AUV act like a highly maneuverable vehicle, the rudder_angle_gain was first adjusted to a high value. This produced the desired high course rate of change for small rudder angles, but caused the rudder to oscillate rapidly back and forth when the AUV passed course North from right to left. The AUV would then be stuck near course North with the rudder flapping back and forth. This behavior amounts to a kind of limit cycle instability [22], and is caused by the discontinuity in heading (0 degrees jumps to 360 degrees) when the heading passes through North.

A smaller rudder_angle_gain was used to prevent the rudder oscillation problem. This smaller gain does result in an AUV that has much more sluggish steering than it should. Figure 6.1 illustrates the AUV executing a turn from an initial course North and position (100, 100) to attain a goal (100, 95). The diagram shows that a very wide turn is necessary to execute this maneuver. In some circumstances, the vehicle can end up in a "tail chasing" mode where it is unable to ever reach a point that is close and directly behind it. In this case, it ends up circling helplessly until the simulation is stopped.

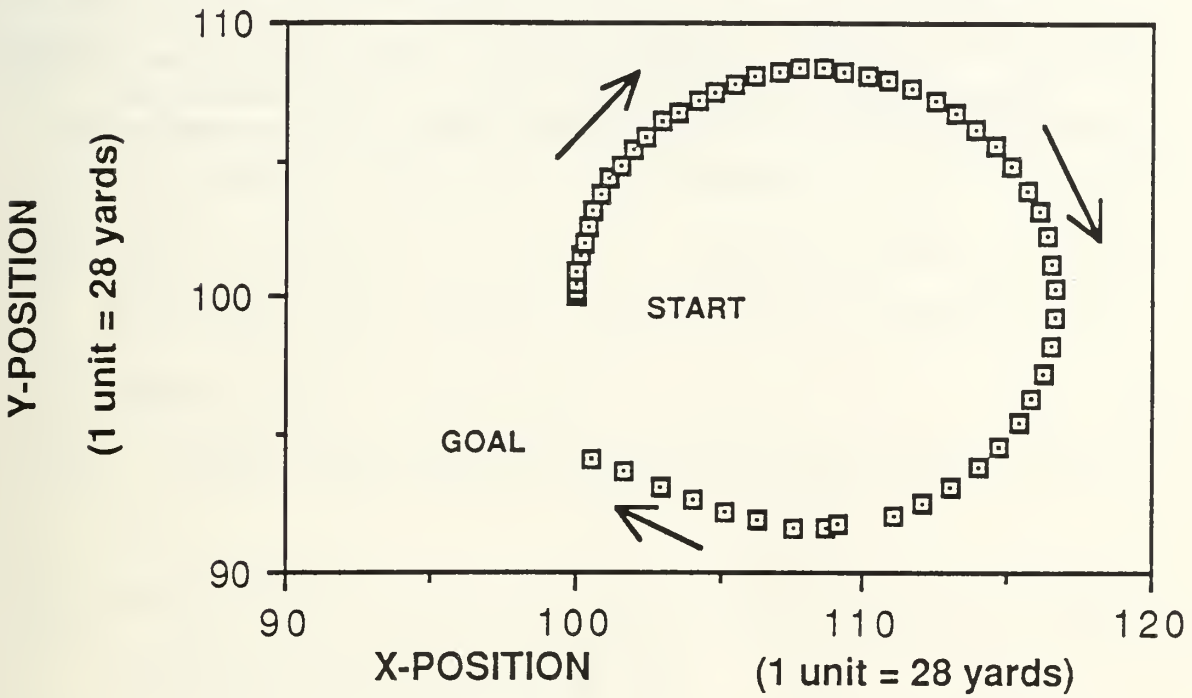


Figure 6.1 AUV MANEUVERIBILITY PLOT SHOWING LARGE TURNING RADIUS RESULTING FROM SMALL RUDDER GAIN

Resolution of this rudder gain problem is an example of how a simulator can be used to solve control problems before they ever can cause trouble in the actual vehicle. However, the solution described above is not the best because of its undesirable affect on maneuverability. This problem has been solved in a more satisfactory way in a previous study of autopilots for ground vehicles [23] by adding or subtracting 360 degrees to azimuth at North crossing to eliminate jumps in value. This technique should be investigated in future simulator studies in order to obtain a more maneuverable AUV.

LIST OF REFERENCES

[1] McTamaney, L., "Mobile Robots Real-Time Intelligent Control," *IEEE Expert,* Winter, 1987, pp. 55.

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

[3] Brady, M., Gerhardt, L., and Davidson, H., "Artificial Intelligence and Robotics," *Robotics and Artificial Intelligence, Series F: Computer and System Sciences,* Springer-Verlag, Vol. 11, 1984, pp. 47.

[4] _, "GATERS Program Moves Forward," *Marine Corps Gazette,* June 1987, p. 8.

[5] Elson, B., "U.S. Army Considers Aquila RPV Ready," *Aviation Week and Space Technology,* November 29, 1982, pp. 54-57.

[6] _, "Army Completing Development Tests for Lockheed Aquila Development Tests," *Aviation Week and Space Technology,* April 28, 1986, pp. 85-88.

[7] _, "Development Sciences Prepares Skyeye for Army Competition," *Aviation Week and Space Technology,* April 28, 1986, pp. 68-70.

[8] Sorrels, C., *U.S. Cruise Missile Programs Development, Deployment and Implications for Arms Control,* McGraw-Hill, Inc., 1983, pp. 133-142.

[9] Nitao, J. and Parodi, A., "A Real-Time Reflexive Pilot for an Autonomous Land Vehicle," *IEEE Control Systems Magazine,* February 1986, pp. 14-23.

[10] Lowrie, J., *The Autonomous Land Vehicle Program - Seventh Quarterly Report,* Martin Marietta Denver Aerospace, Denver, Colorado, November 1987, pp. 4-30.

[11] Freund, J., *New Technologies in U.S. Navy Undersea Vehicles,* Naval Ocean Sea System Command, San Diego, California, February 1987, pp. 2-8.

[12] Bulter, B. and Maryka, S., "Evolution of the Dolphin Multivehicle Control System," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology,* Vol. 5, June 1987, 23-28.

[13] Jalbert, J., "Low Level Architecture for the New Eave Vehicle," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology,* Vol. 5, June 1987, pp. 238-243.

[14] Shevenell, M., "Hardware & Software Architectures for Realizing a Knowledge Based System on EAVE," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology,* Vol. 5, June 1987, pp. 220-228.

[15] Pugh, G. and Krupp, J., "The Control of Autonomous Underwater Vehicles Through a Hierarchical Structure of Value Priorities," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology,* Vol. 5, June 1987, pp. 477-501.

[16] _, *IRIS User's Guide,* Silicon Graphics, Inc., Mountain View, California, 1986.

[17] _, *Texas Instrument Explorer User's Manual,* Texas Instrument Inc., Austin, Texas, 1985.

[18] Boncal, R., *A Study of Model Based Maneuvering Controls for Autonomous Underwater Vehicles,* M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1987.

[19] Nelson, A. and McConkle, C., *A Prototype Simulation System for Combat Vehicle Coordination and Motion Visualization,* M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1988.

[20] Barrow, T., *Distributed Computer Communications in Support of Real-Time Visual Simulations,* M.S. Thesis, Naval Postgraduate School, Monterey, California, June, 1988.

[21] Barr, A. and Feigenbaum, E., *The Handbook of Artificial Intelligence,* William Kaufmann, Inc., Los Altos, California, 1981, pp. 58-64.

[22] Perkin, W. and Cruz, J., *Engineering of Dynamic Systems,* John Wiley and Sons, New York, 1969.

[23] Dolezal, M., *A Simulation Study of a Speed Control System for Autonomous On-Road Operation of Automotive Vehicles,* M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1987.

APPENDIX A - MISSION TEMPLATES USED IN THIS STUDY

All of the AUV mission software written for this study is represented by the mission templates in this Appendix. A pictorial explanation of a specific instance of each generic mission is provided to aid the reader's understanding. The step numbers in the "Mission Format" portion of each mission template correspond to the step numbers on the diagrams for each mission.

AUV MISSION TEMPLATE:

NAME: Bottom Charting

PURPOSE: Perform a bottom charting of a given volume of ocean starting at a given position.

DURATION: Variable

TRANSIT DEPTH: 300 ft

TRANSIT SPEED: 5 kts

ON STATION TIME: To Complete Charting Pattern

ACTION ON STATION: Execute charting pattern, record all fathometer information on magnetic tape.

MISSION FORMAT:

1. Dive and transit to search position.
2. When ordered search position is reached, start fathometer recording equipment.
3. Execute charting pattern.
4. When charting pattern is completed, return to transit depth and speed. Then transit back to starting position.
5. Come to periscope depth and circle at starting postion for battery recharge, data retrieval and further mission assignment.

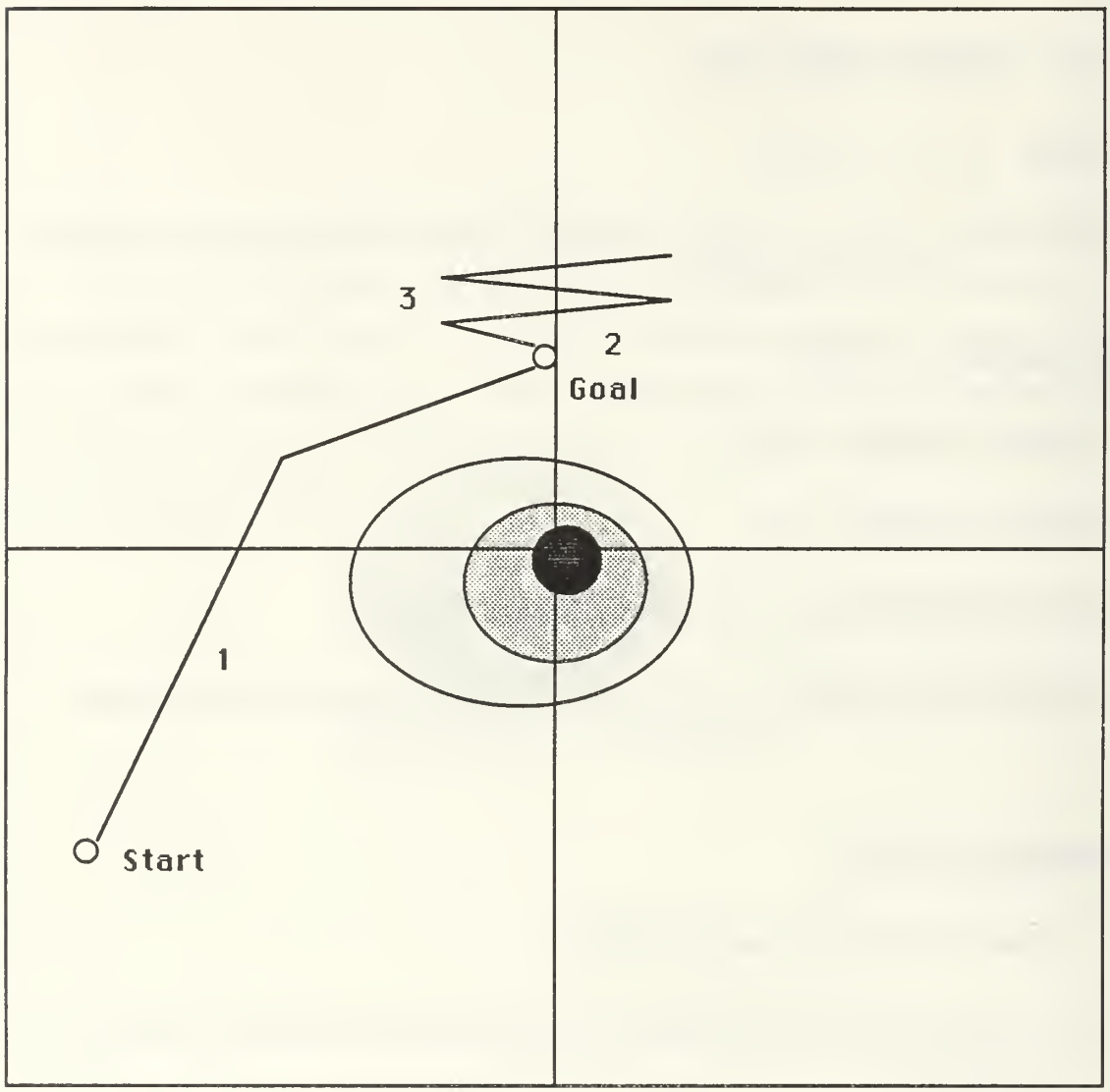


Figure A.1 - BOTTOM CHARTING MISSION

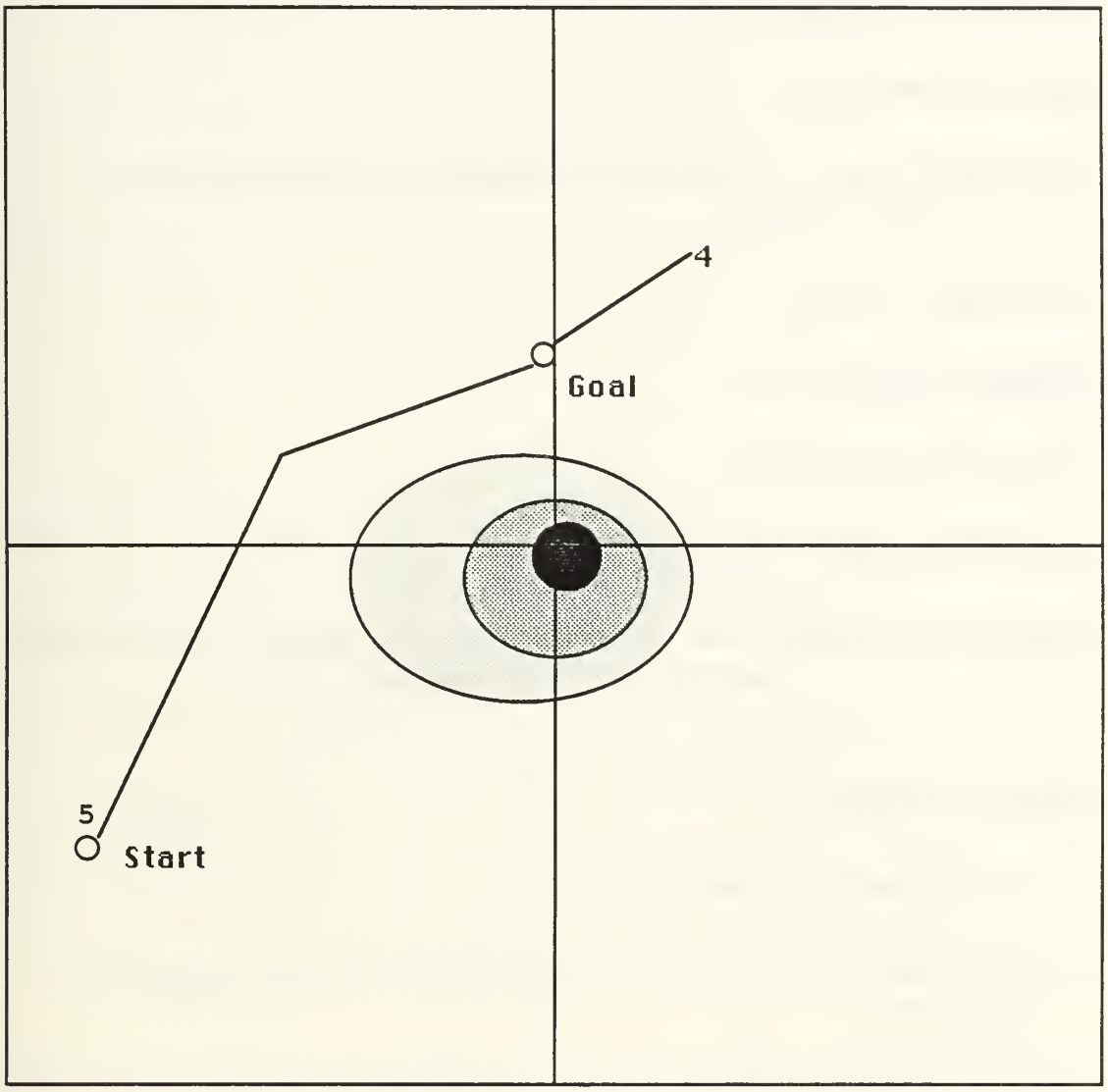


Figure A.2 - BOTTOM CHARTING MISSION

AUV MISSION TEMPLATE:

NAME: Bottom Search

PURPOSE: Perform a sonar search of the ocean bottom starting at a given position.

DURATION: Variable

TRANSIT DEPTH: 300 ft

TRANSIT SPEED: 4 kts

ON STATION TIME: To Complete Search

ACTION ON STATION: Execute search pattern, investigate and photograph all items found on the ocean bottom.

MISSION FORMAT:

1. Dive and transit to search position.
2. When ordered search position is reached, dive to 150 ft above the ocean bottom.
3. Execute search pattern.
4. Localize and photograph all items found on the bottom.
5. When search pattern is completed, return to transit depth and speed. Then transit back to starting position.
6. Come to periscope depth and circle at starting position for battery recharge and further mission assignment.

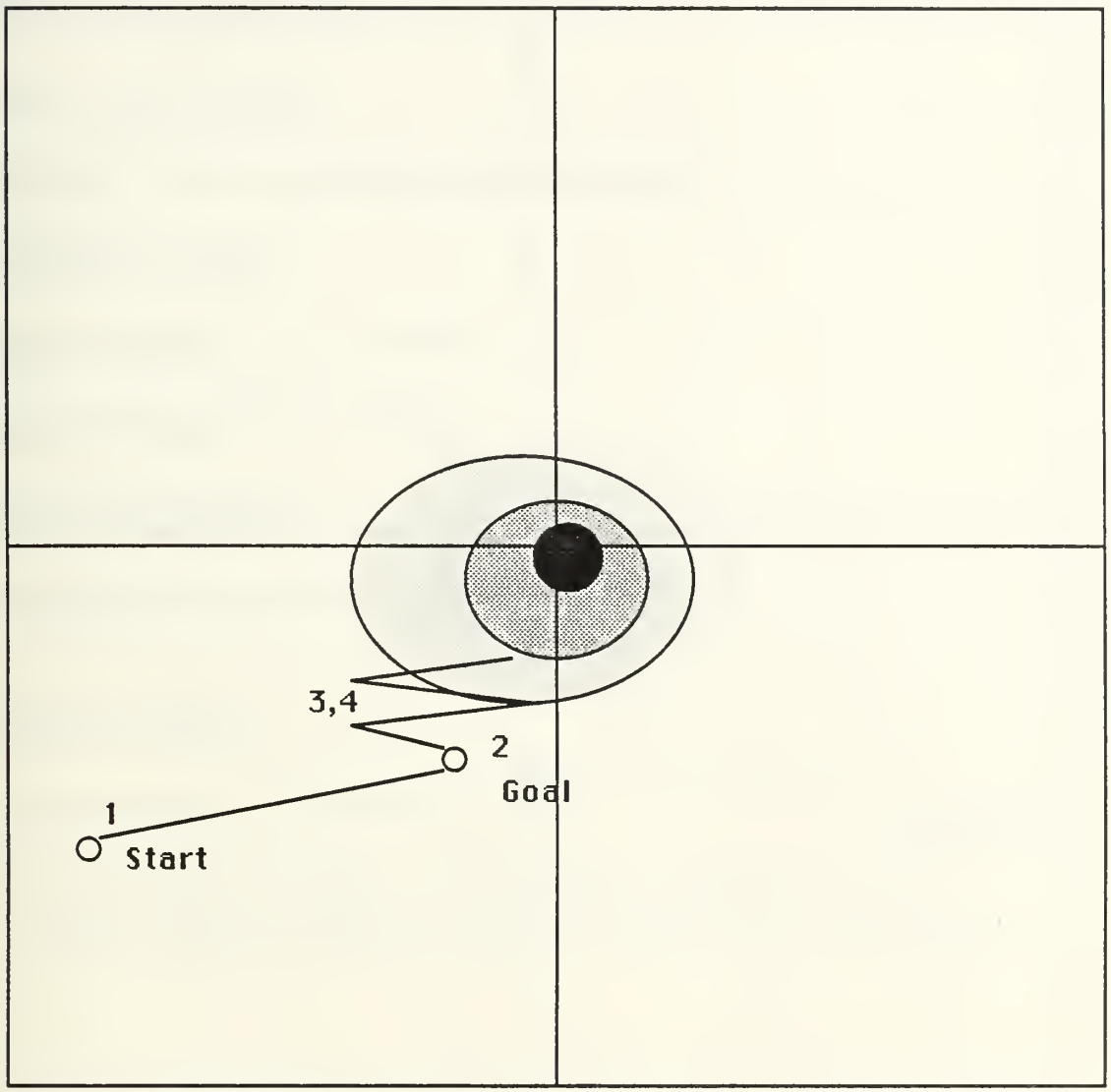


Figure A.3 - BOTTOM SEARCH MISSION

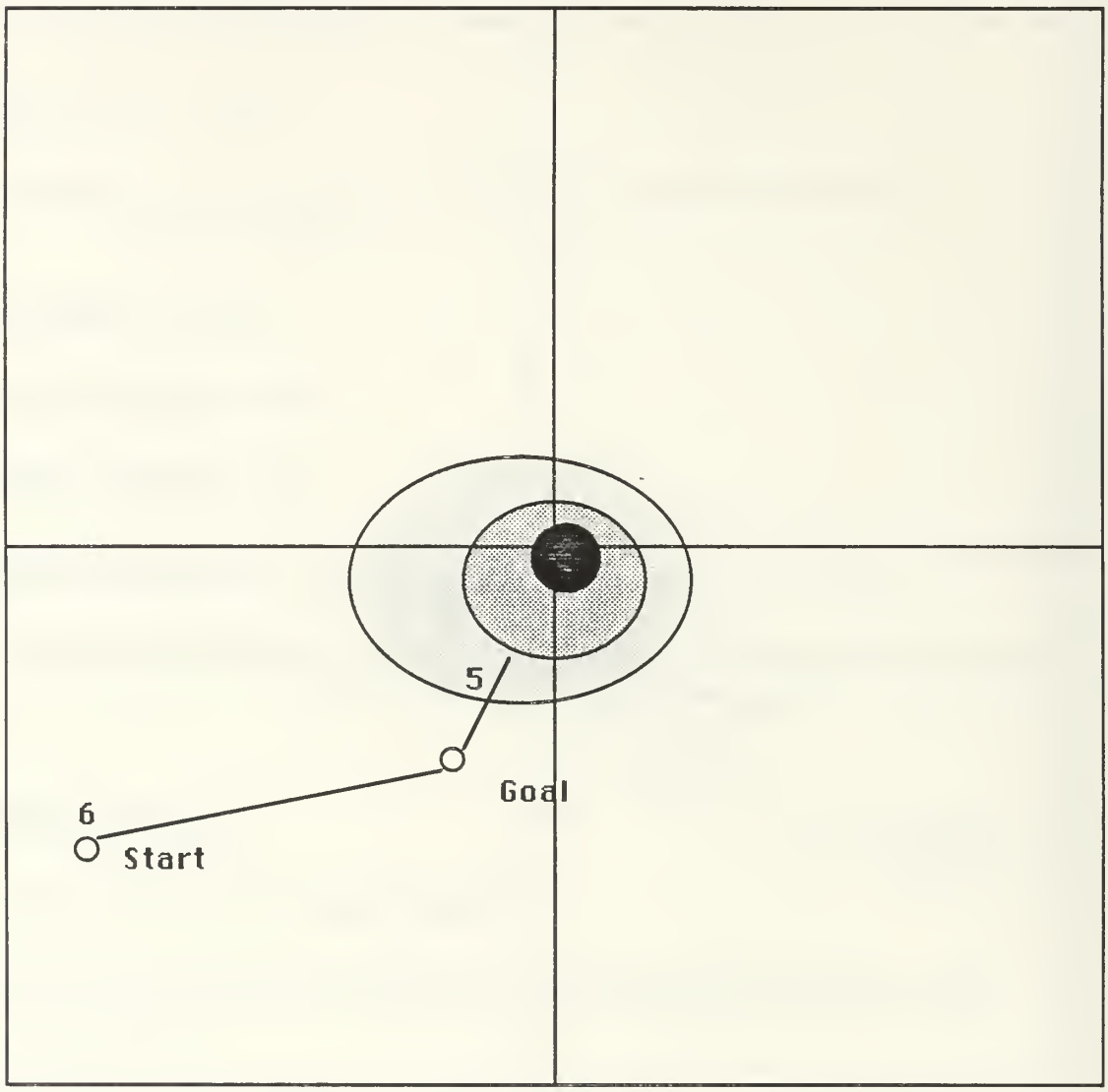


Figure A.4 - BOTTOM SEARCH MISSION

AUV MISSION TEMPLATE:

NAME: Deliver Payload

PURPOSE: Deliver a payload to specified position.

DURATION: Variable

TRANSIT DEPTH: User selectable.

TRANSIT SPEED: User selectable.

ON STATION TIME: N/A

ACTION ON STATION: N/A

MISSION FORMAT:

1. Dive and transit to specified position.

2. Come to periscope depth and circle at the specified position for battery recharge, payload recovery and further mission assignment.

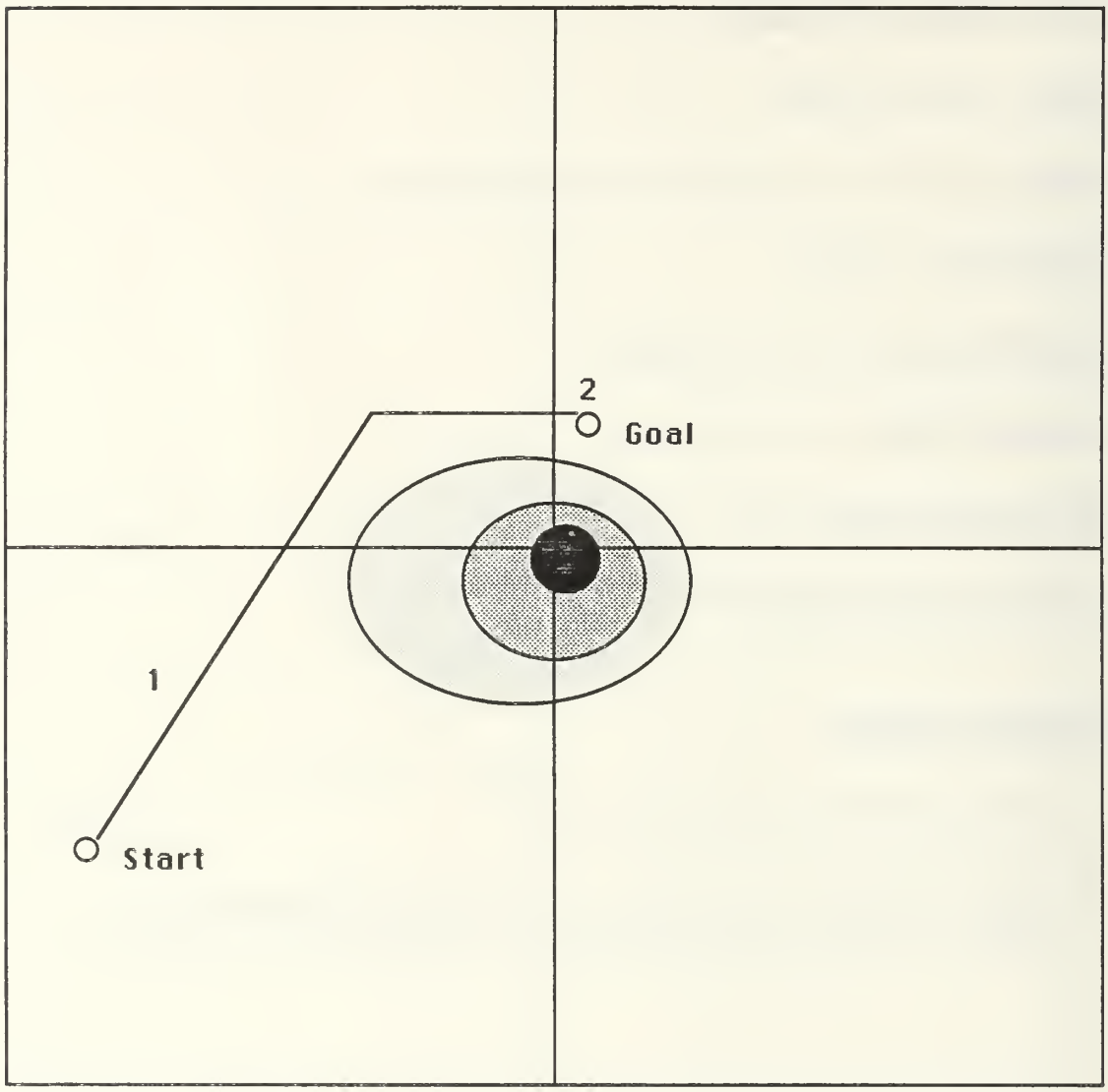


Figure A.5 - DELIVER PAYLOAD MISSION

AUV MISSION TEMPLATE:

NAME: Electronic Recon

PURPOSE: Perform electronic intelligence collection at a given position.

DURATION: Variable

TRANSIT DEPTH: 200 ft

TRANSIT SPEED: 5 kts

ON STATION TIME: Selectable (in minutes)

ACTION ON STATION: Circle at low speed, periscope depth near assigned position with antenna raised. Record all ambient electronic data.

MISSION FORMAT:

1. Dive and transit to collection position.
2. Come to periscope depth.
3. Raise intelligence collection antenna.
4. Circle at periscope depth during on station time, record all electronic signals for on station time.
5. Lower antenna, dive and transit back to starting position.
6. Come to periscope depth and circle at starting postion for battery recharge, data retrieval and further mission assignment.

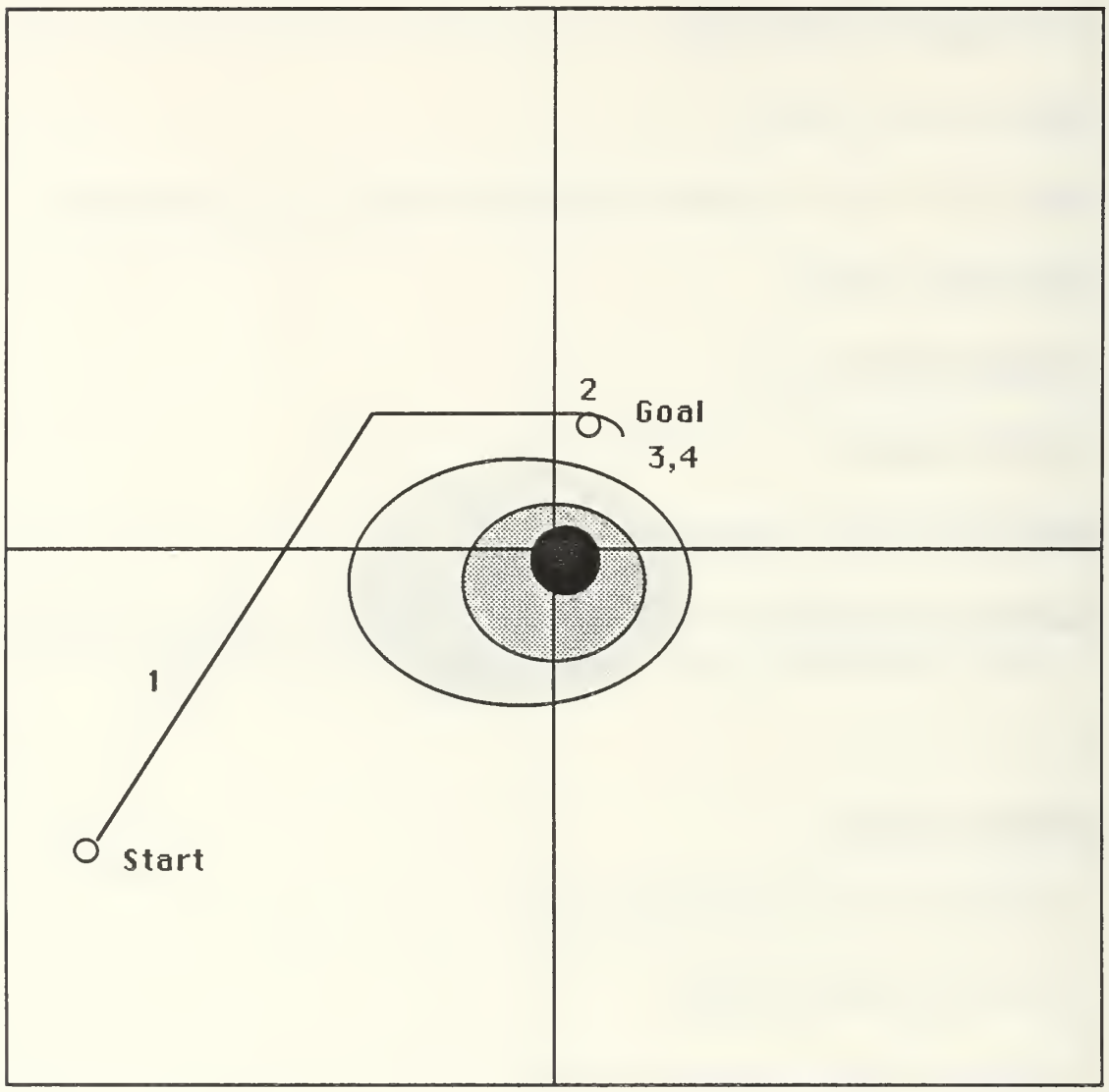


Figure A.6 - ELECTRONIC RECON MISSION

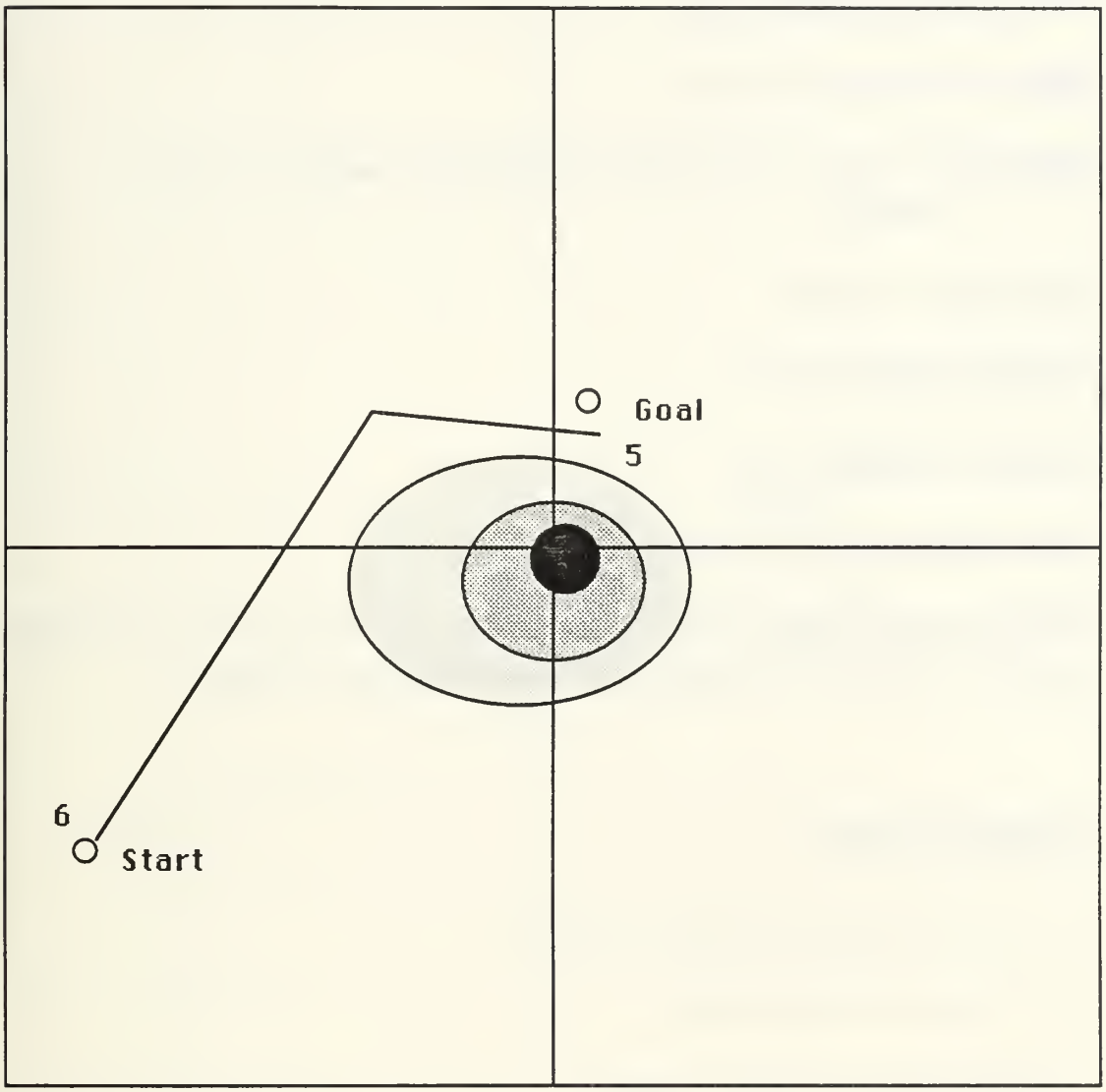


Figure A.7 - ELECTRONIC RECON MISSION

AUV MISSION TEMPLATE:

NAME: Photographic Recon

PURPOSE: Perform photographic intelligence collection at a given position.

DURATION: Variable

TRANSIT DEPTH: 200 ft

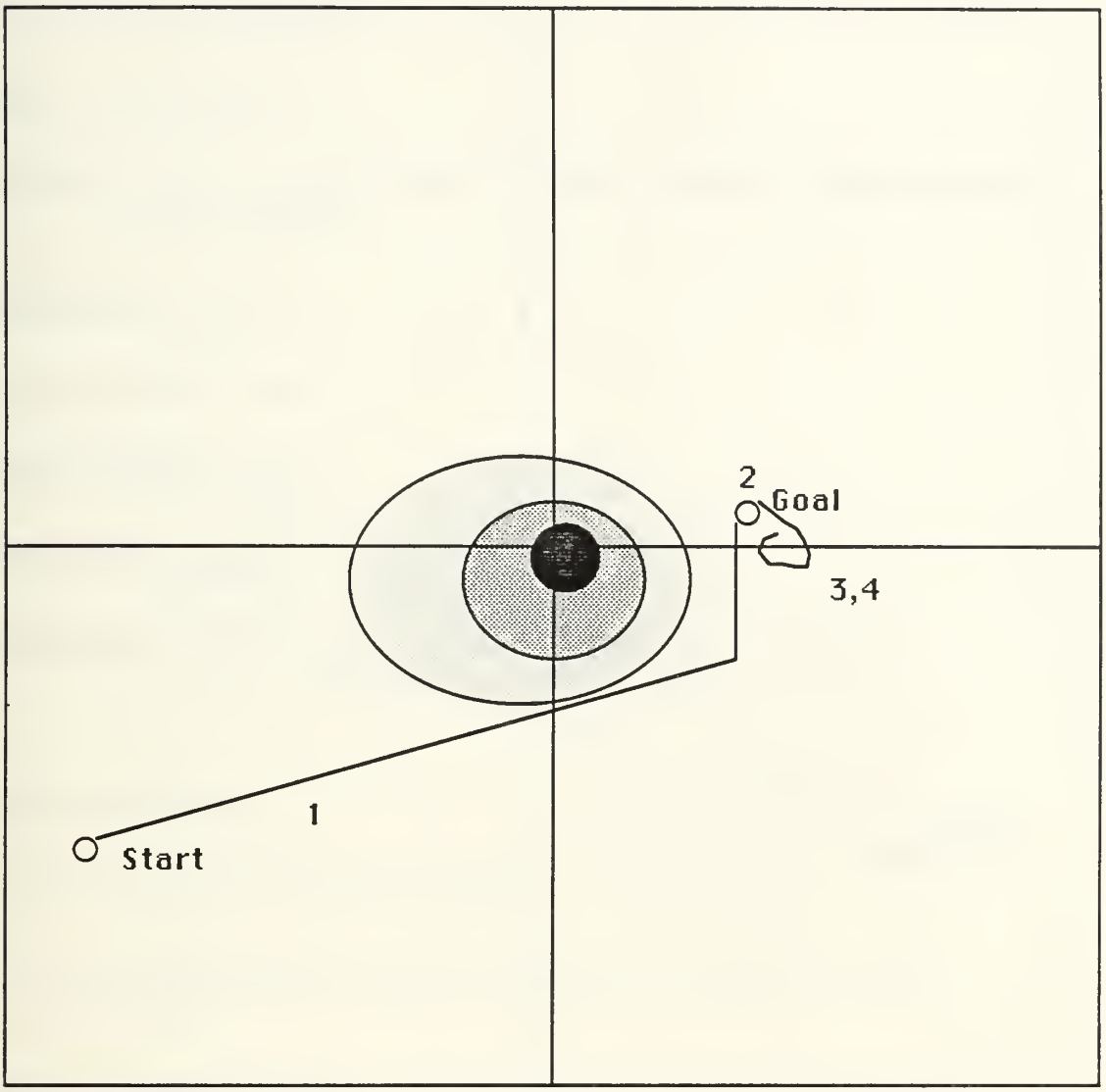
TRANSIT SPEED: 5 kts

ON STATION TIME: Selectable (in minutes)

ACTION ON STATION: Circle at low speed, periscope depth near assigned position with periscope raised. Photograph assigned targets.

MISSION FORMAT:

1. Dive and transit to collection position.
2. Come to periscope depth.
3. Raise periscope, start camera.
4. Circle at periscope depth during on station time, photograph designated target for on station time.
5. Lower periscope, dive and transit back to starting position.
6. Come to periscope depth and circle at starting postion for battery recharge, data retrieval and further mission assignment.



Fugure A.8 - PHOTOGRAPHIC RECON MISSION

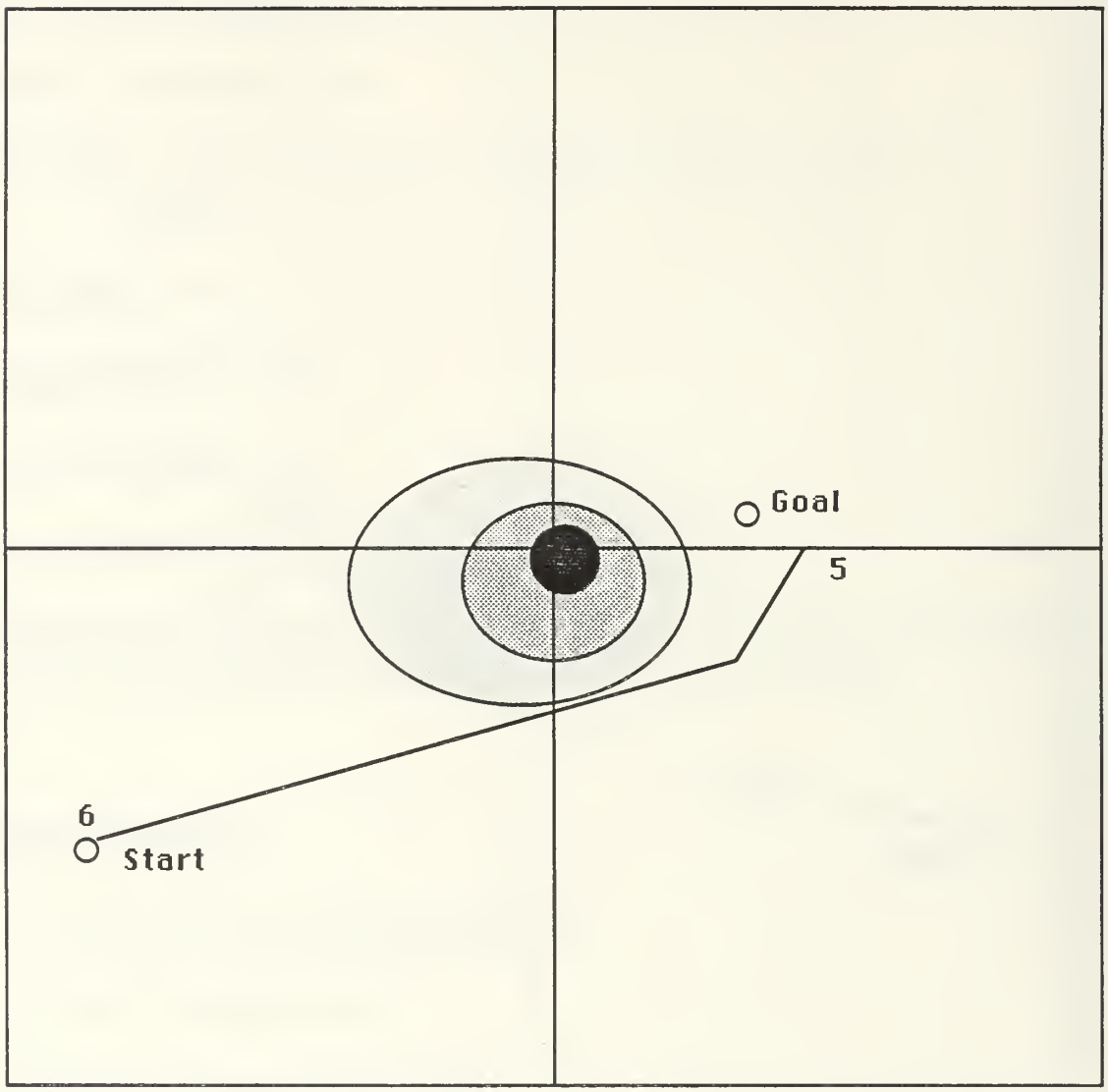


Figure A.9 - PHOTOGRAPHIC RECON MISSION

AUV MISSION TEMPLATE:

NAME: Sonar Search

PURPOSE: Perform a sonar search of a given volume of ocean starting at a given position.

DURATION: Variable

TRANSIT DEPTH: 300 ft

TRANSIT SPEED: 5 kts

ON STATION TIME: To Complete Search

ACTION ON STATION: Execute search pattern, record all sonar information on magnetic tape.

MISSION FORMAT:

1. Dive and transit to search position.

2. When ordered search position is reached, start sonar recording equipment.

3. Execute search pattern.

4. When search pattern is completed, return to transit depth and speed. Then transit back to starting position.

5. Come to periscope depth and circle at starting postion for battery recharge, data retrieval and further mission assignment.

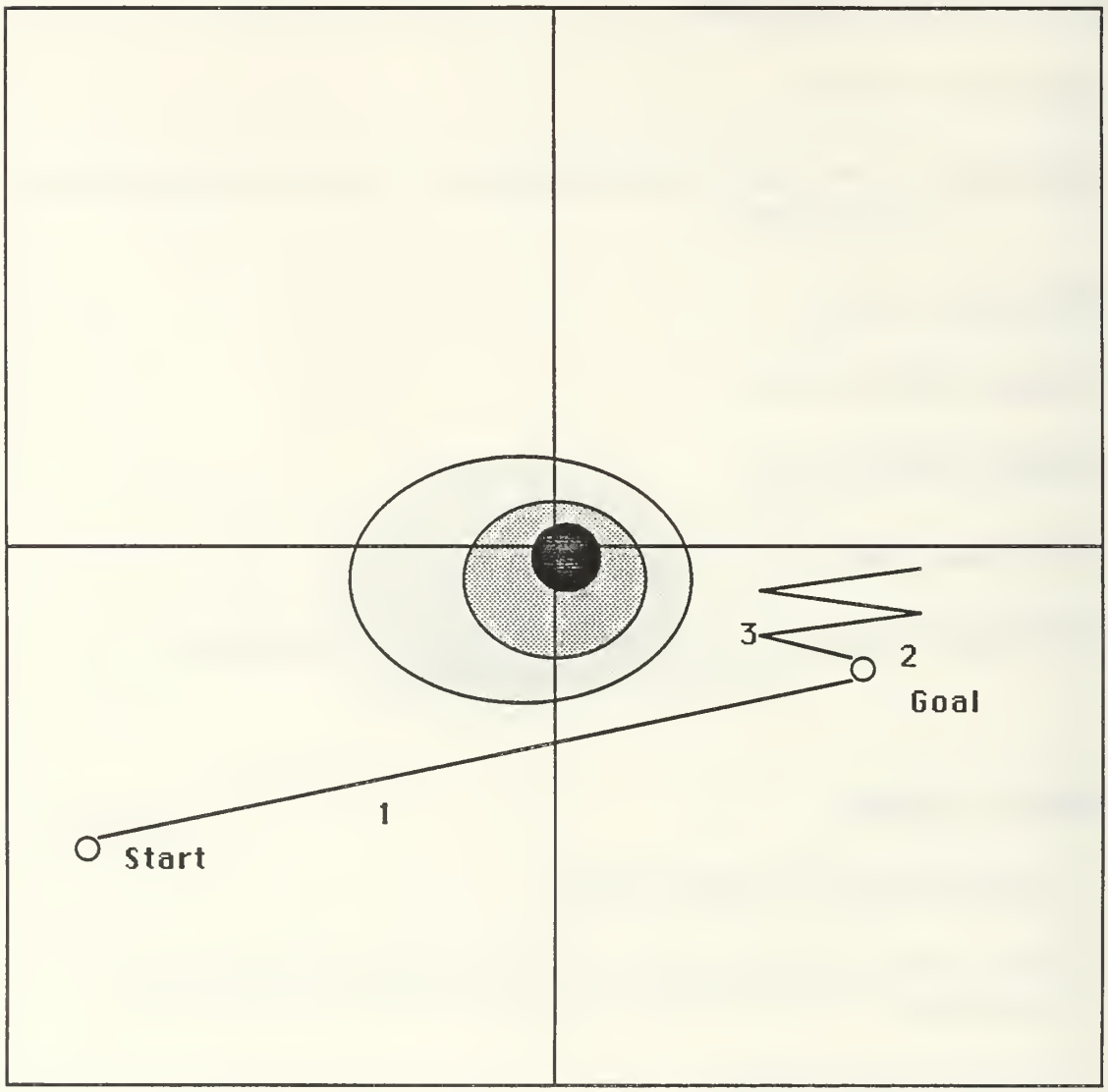


Figure A.10 - SONAR SEARCH MISSION

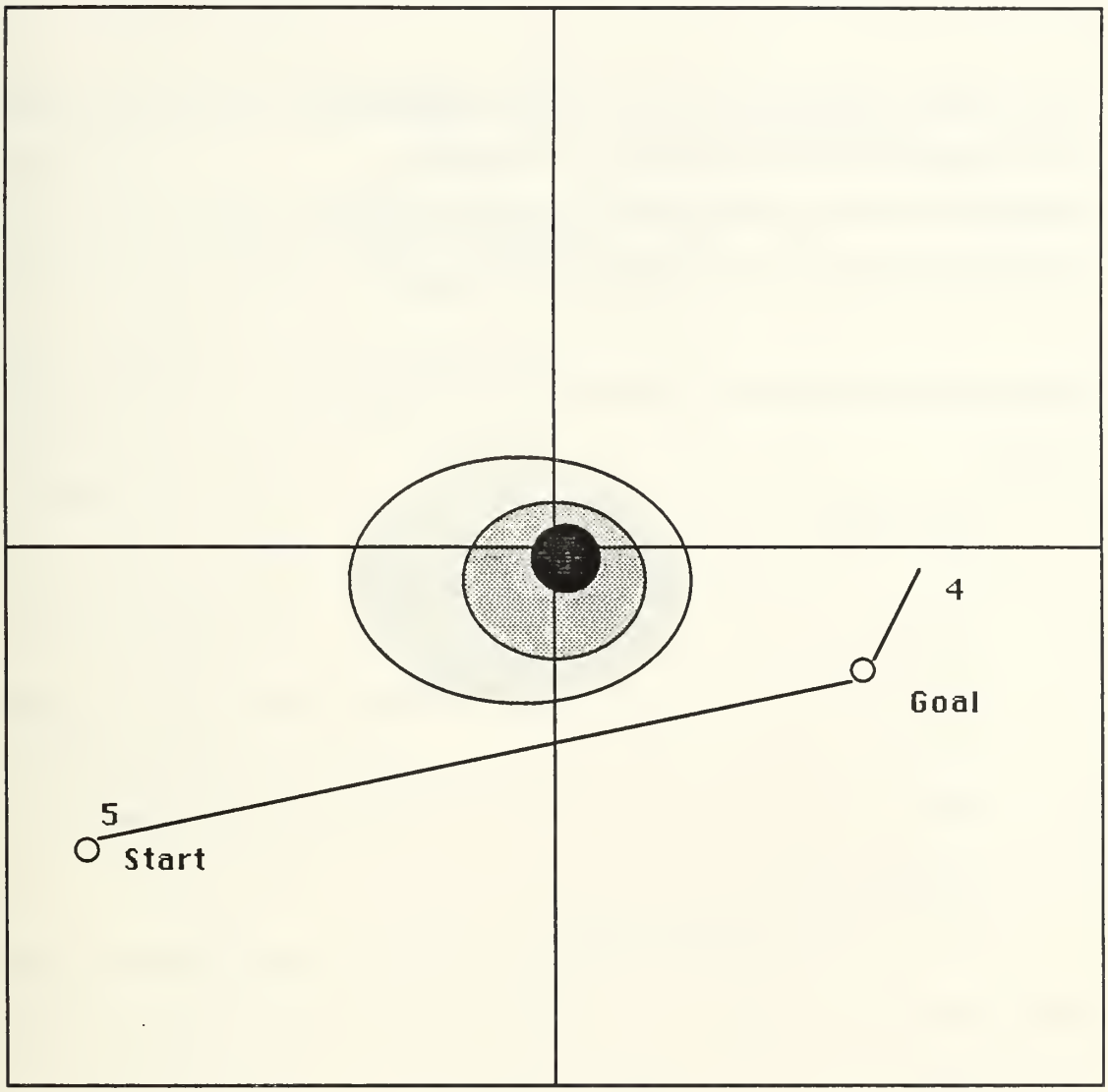


Figure A.11 - SONAR SEARCH MISSION

APPENDIX B - THE SIMULATION PROGRAM CODE

```
;;THIS IS THE PROGRAM IRIS-FLAVOR4.LISP#7.  IT PROVIDES
;;THE NECESSARY SOFTWARE ON THE TI EXPLORER TO
;;COMMUNICATE WITH THE IRIS-2400.  IT MUST BE LOADED
;;IN THE KEE WORLD WITH THE KNOWLEDGE BASE SUB.U

(defmacro loopfor (var init test exp1 &optional exp2 exp3 exp4 exp5)
  `(prog ()
         (setq ,var ,init)
         tag
           ,exp1
           ,exp3
           ,exp4
           ,exp5
           (setq ,var (1+ ,var))
           (if (= ,var ,test) (return t) (go tag))))

(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)
  (catch 'exit
    (dotimes (x (length str) nil)
```

```
    (if (equal (char str x) (char "." 0))
        (throw 'exit 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)))


(defvar *tcp-handler1* (send ip::*tcp-handler* :get-port))
(defvar *tcp-handler2* (send ip::*tcp-handler* :get-port))

(defvar *iris1-port1* 1027)                ; this is the send port
(defvar *iris1-port2* 1026)                ; this is the receive port
(defvar *iris1-address* 3221866502)            ; tcp-ip or internet address
(defvar *iris2-address* 3221866504)            ; look in network configuration
(defvar *unix1-address* 3221866505)
(defvar *unix2-address* )

(defun choose-iris(x)                      ; iris2 is default
 (cond ((equal x 'iris1) (defvar *dest-address* *iris1-address*))
```

```
    ((equal x 'iris3) (defvar *dest-address* *iris3-address*))
    (t          (defvar *dest-address* *iris2-address*))))

(defflavor conversation-with-iris ((talking-port-number   *iris1-port1*)
                                   (listening-port-number  *iris1-port2*)
                                   (talking-port          *tcp-handler1*)
                                   (listening-port        *tcp-handler2*)
                                       (destination          *dest-address*)
                    )
                        ()
                    :gettable-instance-variables
                                :settable-instance-variables
                                :initable-instance-variables)

(defmethod (conversation-with-iris :initiate-the-conversation) ()
 (progn
 (send talking-port :open
    :active                 ; tcp will begin the procedure to establish
                        ; connection  (default  vs :passive)
    talking-port-number         ; port number of destination host
    destination                 ; machine name or address if blank and
                        ; in :passive mode local machine waits for
                        ; connection
    30)                   ; set max seconds  before read request times out

 (send listening-port :open
   :active                          ;:passive
   listening-port-number
```

```
    destination
    30)
 '"A conversation with the iris machine has been established"))

(defmethod (conversation-with-iris :re-use-iris) ()
 (setq *tcp-handler1* (send ip::*tcp-handler* :get-port)
       *tcp-handler2* (send ip::*tcp-handler* :get-port)
       talking-port *tcp-handler1*
       listening-port *tcp-handler2*))

(defmethod (conversation-with-iris :get-an-object-from-iris) ()
 (let* ((typebuffer   " ")
        (lengthbuffer "    ")
      (buffer      " ")
        (buffer-length 1))
  (progn
    (send listening-port :receive
          typebuffer
          buffer-length
          30
          :wait)

        (send listening-port :receive
          lengthbuffer
          4
          30
          :wait)

        (setq buffer-length (convert-string-to-integer lengthbuffer))
```

```
        (setq buffer (make-string buffer-length :initial-element (character 32)))

        (send listening-port :receive
            buffer
            buffer-length
            30
            :wait)
        (cond ((equal typebuffer "I") (convert-string-to-integer buffer))
            ((equal typebuffer "R") (convert-string-to-real    buffer))
            ((equal typebuffer "C") buffer)
            (t nil)))))

(defvar *step-var* 0)

(defmethod (conversation-with-iris :send-iris-an-object) (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) '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) 'float) "R")
                                    ((equal (type-of object) 'string) "C")
                                    (t "C")))
```

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

  (progn
    (send talking-port :send
          typebuffer
          1
          nil
          nil)
                        (if (= (length lengthbuffer) 4)
          (send talking-port :send
              lengthbuffer
              4
              nil
              nil)
        (progn
          (loopfor *step-var* (length lengthbuffer) 4
              (send talking-port :send "0" 1 nil nil))
              (send talking-port :send lengthbuffer (length lengthbuffer) nil nil)
      ))
      (send talking-port :send
          buffer
          buffer-length
          t
          nil))))

(choose-iris 2)

(defmethod (conversation-with-iris :stop-talking-to-iris) ()
 (progn (send talking-port :close) (send listening-port :close)))
```

```
(setq talk (make-instance 'conversation-with-iris))

(defun start-con () (send talk :initiate-the-conversation))

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

(defun restart () (send talk :re-use-iris))

(defun get_data ()
  (send talk :get-an-object-from-iris))

(defun send_float (float)
  (send talk :send-iris-an-object float))

(defun send_string (string)
  (send talk :send-iris-an-object string))
```

```
;;THIS IS PROGRAM AP3.LISP#112.  IT PROVIDES THE GUIDANCE LEVEL
;;CODE WRITTEN IN COMMON LISP.  VARIOUS AUVS MISSIONS ARE CODED
;;IN THIS PROGRAM.  THIS PROGRAM MUST BE LOADED WITH THE SUB.U
;;KNOWLEDGE BASE TO RUN AUVS MISSIONS.

(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)
  (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 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))

(defun transit_back (autodepth autospeed)
    (setq path rev_path)
    (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))))
    (get_closest_range sonar_contacts)
    (princ closest_contact_range)
    (terpri))

(defun get_closest_range (contacts)
  (setq closest_contact_range (caar contacts))
  (do ((contact_list contacts
                (cdr contact_list))
    (list_length (length contacts)
```

```
            (1- list_length)))
        ((< list_length 1))
        (if (< (caar contact_list) closest_contact_range)
            (setq closest_contact_range (caar contact_list)))))


(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 on the first leg is :")
    (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)))
        ((and (> 5 closest_contact_range) (< autodepth 100))
                          (send_float 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 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 load_files ()
  (load "aries: macpher; best.xfasl")
  (load "aries: macpher; iris-flavor4.lisp"))


(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))))))
```

```
;;THIS IS THE PROGRAM BEST.LISP#4.  IT USED BY THE GUIDANCE
;;LEVEL SOFTWARE TO PERFORM A BEST FIRST SEARCH.  THIS
;;PROGRAM IS WRITTEN IN COMMON LISP.

(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)
   ((> 20 (distance x y 205 205)) 50)
   (t (* 0.2 (square (distance x y 200 200))))))

(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:Common-Lisp; Syntax:Common-lisp; Package:KEE; Base:10. -*-
;;THIS IS THE PROGRAM SUB.U.  IT IS THE KEE KNOWLEDGE BASE
;;FOR THE USER TO SELECT THE DESIRED MISSION.  THIS
;;KNOWLEDGE BASE MUST BE LOADED FIRST IN KEE, THEN LOAD AP3.LISP,
;;BEST.LISP AND IRIS-FLAVOR4.LISP TO RUN THE AUVS.

(SUB

("MACPHER" "11-9-87 9:45:56" "MACPHER" "12-1-87 13:02:16")
 NIL

(KNOWLEDGEBASES)
 NIL
 ()
 ( (KBMETHODFILE ("SUB"))

(KBSIZE (14))

(KEE.DEVELOPMENT.VERSION.NUMBER (0))

(KEE.MAJOR.VERSION.NUMBER (3))

(KEE.MINOR.VERSION.NUMBER (0))

(KEE.PATCH.VERSION.NUMBER (159))

(KEEVERSION (KEE3.0)) ))


(ELECTRONIC
 ("MACPHER" "11-9-87 9:47:48" "MACPHER" "11-21-87 9:37:18")
 (INTELL.GATHERING)
 ((CLASSES GENERICUNITS))
```

```
 NIL ()
 ((LOAD.LISP.FILES ((LAMBDA (THISUNIT)
              (LOAD "aries: macpher; ap3.lisp")
              (LOAD_FILES))) METHOD
(#[Unit: METHOD KEEDATATYPES]) NIL ((COMMENT
                        "loads the necessary lisp files")))
  (START.ELECT.RECON.MISSION
((LAMBDA (THISUNIT)
   (PRINC "Enter x position of survey")
   (TERPRI)
   (SETQ X1 (READ))
   (PRINC "Enter y position of survey")
   (TERPRI)
   (SETQ Y1 (READ))
   (PRINC "Enter the number of minutes on station")
   (TERPRI)
   (SETQ TIME (READ))
   (ELECT_RECON_MISSION X1 Y1 TIME)))
 METHOD (#[Unit: METHOD KEEDATATYPES]) NIL ((COMMENT
                          "electronic recon mission")))
  ))

(PHOTOGRAPHIC
 ("MACPHER" "11-9-87 9:47:48" "MACPHER" "12-1-87 12:11:55")
 (INTELL.GATHERING)
 ((CLASSES GENERICUNITS))
 NIL ()
 ((START.PHOTO.RECON.MISSION
```

```
 ((LAMBDA (THISUNIT)
   (PRINC "Enter x position of survey")
   (TERPRI)
   (SETQ X1 (READ))
   (PRINC "Enter y position of survey")
   (TERPRI)
   (SETQ Y1 (READ))
   (PRINC "Enter the time on station")
   (TERPRI)
   (SETQ TIME_ON_STATION (READ))
   (PRINC "Enter true bearing to train periscope")
   (TERPRI)
   (SETQ PERISCOPE_BEARING (READ))
   (PHOTO_RECON_MISSION X1 Y1 TIME_ON_STATION PERISCOPE_BEARING)))
  METHOD (#[Unit: METHOD KEEDATATYPES]) NIL ((COMMENT
                              "initiates an photographic recon mission")))
  ))

(INTELL.GATHERING
 ("MACPHER" "11-9-87 9:47:17" "MACPHER" "11-19-87 18:14:41")
 (MISSIONS)
 ((CLASSES GENERICUNITS))
 NIL ()
 ())

(UNDER.ICE
 ("MACPHER" "11-9-87 9:48:17" "MACPHER" "11-9-87 9:48:17")
 (CHART.DATA.GATHERING)
```

((CLASSES GENERICUNITS))

NIL ()

())

(BOTTOM.CHARTING

("MACPHER" "11-9-87 9:48:17" "MACPHER" "11-19-87 18:15:55")

(CHART.DATA.GATHERING)

((CLASSES GENERICUNITS))

NIL ()

())

(CHART.DATA.GATHERING

("MACPHER" "11-9-87 9:47:15" "MACPHER" "11-19-87 18:15:12")

(MISSIONS)

((CLASSES GENERICUNITS))

NIL ()

())

(MINE.WARFARE

("MACPHER" "11-19-87 18:17:09" "MACPHER" "11-19-87 18:17:09")

(COVERT)

((CLASSES GENERICUNITS))

NIL ()

())

(TRANSPONDER.DELIVERY

("MACPHER" "11-19-87 18:17:09" "MACPHER" "11-19-87 18:17:09")

(COVERT)

```
 ((CLASSES GENERICUNITS))
 NIL ()
 ())

(MISSIONS
 ("MACPHER" "11-9-87 9:46:24" "MACPHER" "11-9-87 9:47:17")
 NIL
 ((CLASSES GENERICUNITS))
 "missions for unmanned sub" ()
 ())

(BOTTOM.SEARCH
 ("MACPHER" "11-19-87 18:17:38" "MACPHER" "11-21-87 8:58:29")
 (SEARCH)
 ((CLASSES GENERICUNITS))
 NIL ()
 ((START.BOTTOM.SEARCH.MISSION
((LAMBDA (THISUNIT)
  (PRINC "Enter the x position of the bottom search")
  (TERPRI)
  (SETQ X1 (READ))
  (PRINC "Enter the y position of the bottom search")
  (TERPRI)
  (SETQ Y1 (READ))
  (PRINC "Enter the search speed")
  (TERPRI)
  (SETQ SEARCH_SPEED (READ))
  (BOTTOM_SEARCH_MISSION X1 Y1 SEARCH_SPEED)))
```

```
METHOD (#[Unit: METHOD KEEDATATYPES]) NIL ((COMMENT
                        "starts the bottom search mission")))
 ))

(SONAR.SEARCH
 ("MACPHER" "11-19-87 18:17:38" "MACPHER" "11-20-87 17:09:59")
 (SEARCH)
 ((CLASSES GENERICUNITS))
 NIL ()
 ((LOAD.LISP.FILES ((LAMBDA (THISUNIT)
          (LOAD "aries: macpher; ap3.lisp")
          (LOAD_FILES))) METHOD
(#[Unit: METHOD KEEDATATYPES]) NIL ((COMMENT
                    "loads lisp files")))
  (START.SONAR.SEARCH.MISSION
((LAMBDA (THISUNIT)
  (PRINC "Enter x position of search")
  (TERPRI)
  (SETQ X1 (READ))
  (PRINC "Enter y position of search")
  (TERPRI)
  (SETQ Y1 (READ))
  (PRINC "Enter search depth")
  (TERPRI)
  (SETQ SEARCH_DEPTH (READ))
  (PRINC "Enter search speed")
  (TERPRI)
  (SETQ SEARCH_SPEED (READ))
```

```
  (SONAR_SEARCH_MISSION X1 Y1 SEARCH_DEPTH SEARCH_SPEED)))
 METHOD (#[Unit: METHOD KEEDATATYPES]) NIL ((COMMENT
                              "executes a sonar search mission")))
 ))

(SEARCH
 ("MACPHER" "11-19-87 18:16:36" "MACPHER" "11-19-87 18:16:37")
 (MISSIONS)
 ((CLASSES GENERICUNITS))
 NIL ()
 ())

(DELIVER.PAYLOAD
 ("MACPHER" "11-19-87 18:29:19" "MACPHER" "11-20-87 17:11:01")
 (COVERT)
 ((CLASSES GENERICUNITS))
 NIL ()
 ((LOAD.LISP.FILES ((LAMBDA (THISUNIT)
            (LOAD "aries: macpher; ap3.lisp")
            (LOAD_FILES))) METHOD
(#[Unit: METHOD KEEDATATYPES]) NIL ((COMMENT
                        "loads all lisp files")))
 (START.PAYLOAD.DELIVERY.MISSION
((LAMBDA (THISUNIT)
  (PRINC "Enter x-position for delivery")
  (TERPRI)
  (SETQ X1 (READ))
  (PRINC "Enter y-position for delivery")
```

```
   (TERPRI)
   (SETQ Y1 (READ))
   (PRINC "Enter transit depth")
   (TERPRI)
   (SETQ TRANSIT_DEPTH (READ))
   (PRINC "Enter transit speed")
   (TERPRI)
   (SETQ TRANSIT_SPEED (READ))
   (DELIVER_PAYLOAD_MISSION X1 Y1 TRANSIT_DEPTH TRANSIT_SPEED)))
 METHOD (#[Unit: METHOD KEEDATATYPES]) NIL
 ((COMMENT
  "delivers a payload to a designated position")))
  ))

(COVERT
 ("MACPHER" "11-19-87 18:16:37" "MACPHER" "11-19-87 18:16:37")
 (MISSIONS)
 ((CLASSES GENERICUNITS))
 NIL ()
 ())

KBEnd
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5100	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	Curricular Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor Robert B. McGhee, Code 52Mz Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	9
7.	Professor Michael J. Zyda, Code 52Zk Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1
8.	United States Military Academy Department of Geography & Computer Science ATTN: Major R. F. Richbourg West Point, New York 10996-1695	1

9.	Naval Ocean System Center Ocean Engineering Division (Code 94) ATTN: Paul Heckman San Diego, California 92152-5000	1
10.	Department Chairman, Code 69Hy Mechanical Engineering Department Naval Postgraduate School Monterey, California 93943-5004	1
11.	Professor D.L. Smith, Code 69Sm Mechanical Engineering Department Naval Postgraduate School Monterey, California 93943-5004	1
12.	Professor R. Christi, Code 62Cx Electrical and Computer Engineering Department Naval Postgraduate School Monterey, California 93943-5004	1
13.	Naval Coastal System Center Navigation, Guidance and Control Branch ATTN: G. Dobeck Panama City, Florida 32407-5000	1
14.	Russ Werneth, Code u25 Naval Surface Warfare Center White Oak, Maryland 20910	1
15.	HQDA Artificial Intelligence Center ATTN: DACS-DMA, LTC. Anthony Anconetoni The Pentagon Room 1D659 Washington, D.C. 20310-0200	1
16.	RADM G. Curtis, Code PMS-350 Naval Sea System Command Washington, D.C. 20362-5101	1
17.	Mr. and Mrs. D. L. MacPherson 86 Brent Street Albany, New York 12205	1
18.	Mr. and Mrs. D. L. MacPherson,Jr. 3095 Marina Dr. #31 Marina, California 93933	1

19.	Director of Research Administration Code 012 Naval Postgraduate School Monterey, California 93943	1
20.	Center for Naval Analyses 4401 Ford Ave. Alexandria, VA 22302-0268	1

Thesis
M26946
c.1

MacPherson
A computer simulation study of rule-based control of an Autonomous Underwater Vehicle.