1

Remote Monitoring and Control of Autonomous Underwater Vehicles

by

John H. I. Kim

B.S., General Engineering
Harvey Mudd College (1991)

Submitted to the Department of Ocean Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Ocean Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feb 1997

Author ..... [illegible] ....
Department of Ocean Engineering
January 17, 1997

Certified by ... [illegible] ....
John J. Leonard
Assistant Professor of Ocean Engineering
Thesis Supervisor

Accepted by .... [illegible] ....
J. Kim Vandiver
Chairman, Departmental Committee on Graduate Students

Remote Monitoring and Control of Autonomous Underwater Vehicles

by

John H. I. Kim

Submitted to the Department of Ocean Engineering
on January 17, 1997, in partial fulfillment of the
requirements for the degree of
Master of Science in Ocean Engineering

Abstract

Oceanographic data gathering techniques are moving away from simple sensors completely under the control of the user, towards smart systems such as autonomous underwater vehicles (AUVs) that are capable of operating without user intervention. However, artificial intelligence cannot yet match human intelligence, and the ability to remotely monitor and control these otherwise autonomous systems throughout the data collection process is highly desirable. Communications links to these underwater systems via acoustic modems and the Internet are becoming increasingly viable. This thesis investigates the problems arising from combining these developing technologies for oceanographic data-gathering purposes.

Towards the goal of remotely monitoring and controlling an AUV during a mission, this thesis explores various methods of vehicle control, mission reconfiguration, data transmission, data robustness, interconnectivity with the Internet, and communication with and between multiple AUVs. We have run tests using laboratory simulation and over 25 field experiments in the Charles River and Buzzard's Bay with MIT Sea Grant's Odyssey II AUV, using a pair of radio modems to simulate the acoustic link. Our primary research accomplishment has been to demonstrate control of the Odyssey II during a Charles River mission from a workstation located on the Internet back at the MIT Sea Grant computer laboratory.

Thesis Supervisor: John J. Leonard
Title: Assistant Professor of Ocean Engineering

Acknowledgments

The author would like to thank the following people without whom this thesis would not have been possible.

Mom and Dad for supporting my decision to go into engineering, even though they wanted me to go into medicine.

Chrys Chryssostomidis for heading and directing the MIT Sea Grant Laboratory where this research took place.

James Bellingham, for founding the MIT Sea Grant AUV Lab, and spearheading the development of the Odyssey II class AUV.

Seamus Tuohy for his assistance with multicasting protocols on the Internet.

Thomas Consi for introducing me to MIT Sea Grant.

Bradley Moran for his programming assistance with the software produced in this research.

James Bales and Clifford Goudey for a thorough introduction to the Odyssey II AUV, and assistance with hardware used in this research.

Robert Grieve and Joe Ziehler for their invaluable help producing, fixing, and operating the equipment used in this research.

Scott Willcox, Andrew Bennet, Chris Smith, and Claudia Rodriguez for their assistance during field operations and presence as friends.

The rest of the Sea Grant staff, including Thomas Vaneck, Timothy Downes, Janice Ahern, ReRe Quinn, Joe Curcio, Andrea Cohen, and Kathy de Zengotita for keeping the organization running.

And John Leonard for all the reasons outlined above, but most importantly for his guidance and steadfast support throughout the course of my research and writing.

Support for this work was provided by MIT Sea Grant under grant numbers NA90AA-D-SG-424 and NA46-RG-0434, the Office of Naval Research under grant numbers N00014-91-J-1014 and N00014-92-J-1287, and the Naval Undersea Warfare Center under contract N66604-94-C-D057.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Oceanographic data gathering techniques are progressing from traditional hands-on approaches towards remote sensing capabilities requiring little or no operator presence. This paradigm shift brings up the issue of communication with the equipment; since the operator is not physically present, the data and instructions must be transmitted between the operator and equipment.

The ultimate in remote sensing is the autonomous underwater vehicle (AUV), which by carrying a full instrument suite can duplicate many of the capabilities of ship-based operations. This thesis explores the problems of controlling and communicating with AUVs during oceanographic data-gathering missions. Such communications can take the form of acoustic transmissions through the water, radio links, and transmissions over computer networks such as the Internet. These channels are often unreliable and introduce time delays, bandwidth restrictions, and distortions which cause transmission errors.

To further explore the consequences of using these communication channels, we have investigated monitoring and controlling an AUV over slow, unreliable communication links. We have performed experiments with an Odyssey II AUV towing a radio-frequency modem behind it, and connected to the Internet. Our results offer insights into the remote monitoring and control problem, and make many suggestions for future avenues of research.

1.1 Rationale for Remote Control and Monitoring of AUVs

1.1.1 The problem of gathering oceanographic data

In the last two decades, communications technology and its applications have advanced dramatically bringing us into the so called "Information Age." Innovations such as wide-area networks, wireless data links, and robot probes sent to other planets have become commonplace. In contrast, oceanographic data gathering techniques have lagged behind. This is primarily due to the difficulties encountered in communicating through the oceans. Because water is mostly opaque to electromagnetic radiation, much of the equipment and techniques developed for use on land and through air and space are not applicable. In the cases where this opacity is not a factor, such as with satellites and moored buoys, the nature of these systems restrict their usefulness. Buoys are limited to point measurements, while satellites are limited to the top layer of the oceans. Trying to understand oceanographic phenomena solely from these measurements is akin to trying to understand atmospheric weather based only on ground measurements. To fully measure oceanographic phenomena requires gathering data in all three spatial dimensions.

Gathering 3D oceanographic data has always involved a tradeoff between coverage, control, and cost. An ideal oceanographic platform would have high coverage, complete control, and low cost. However, complete control requires the sensing apparatus and control center be one and the same. The Navy's NR-1 research submarine [26] and the Woods Hole Oceanographic Institution's Alvin [28] are examples of this approach, with all sensors and control apparatus (measuring equipment, personnel, life support, etc.) combined into one vehicle. The bulk of this additional apparatus limits the vehicle's mobility, both in the water and between dives, reducing its potential for covering large areas and increasing operating expense. In response, researchers have reduced instrument packages to the bare minimum needed to do the job. Nonessentials such as people, and in some cases measuring and control equipment, were eliminated to facilitate getting the instruments where you wanted them as quickly and cheaply as possible. By giving up control, coverage is improved and cost reduced.

1.1.2 Current methods of gathering oceanographic data

The simplest instruments are lowered on a line and collect a water sample at a predetermined depth. This sample is then measured for salinity (conductivity) and temperature versus depth (CTD). By moving the ship around and obtaining samples at many locations, the researcher can build a rough 3D map of the ocean. While easier, quicker, and in most cases more feasible than sending a person down, this method sacrifices much in the way of feedback and control. No current measurements are possible, location is inexact, the sample is vulnerable to contamination, and most importantly, measurements can't be made until the sample is back aboard the ship. A CTD cast gains in coverage and cost by reducing control — separating the sensing apparatus from the higher control functions responsible for analysis and placement of the sensor. Furthermore, this separation is not complete. The personnel and equipment needed to complement CTD casts must be aboard the ship doing the casts. This adds to the expense and personnel requirements, and often requires scheduling ship time far in advance of the data collection, thus reducing some of the cost gains.

Tethered equipment such as remotely operated vehicles (ROVs) and towed instrument sleds attempt to regain some of the control lost in the simplest sensors described above [40]. By running a communications line through the cable, the tethered vehicle can provide more flexibility and immediacy in its sensor coverage. Such systems offer immediate reports on measurements, and give the researcher some freedom to move the equipment around in the water. This increase in control comes at price of increased bulk and expense. Measuring equipment must now be housed in the vehicle, and the tether design more elaborate than a simple wire cable. In addition, tethered equipment does not eliminate the need for a ship to house the personnel and additional support equipment; ship time for a large research vessel often represents a substantial portion of a budget.

1.1.3 Remote sensing via AUVs as a new way to conduct oceanography

Autonomous underwater vehicles (AUVs) have been proposed as a solution to the tether problem [8, 7]. By eliminating the tether completely, AUVs allow complete mobility and flexibility in sensor coverage, requiring little more hardware than ROVs to accomplish this. Their main drawbacks to date have been power, software, and control. Because AUVs are by definition autonomous, they must be programmed with enough artificial intelligence to observe, react to, and in some cases learn from their environment in order to simply survive, much less accomplish their mission. This questionable robustness and loss of control on the researcher's behalf has long been a stumbling block to their widespread acceptance.

Recent advances in underwater acoustic modem technology and AUVs has opened up a promising alternative. By using the ocean itself as an acoustic communications channel, the mobility and flexibility of an AUV's coverage can be combined with the ROV's controllability and immediate reports of sensor measurements. While this eliminates the cost of the tether, it does not address the cost of the support ship. Work currently being done at the MIT Sea Grant AUV Laboratory has called for an Autonomous Ocean Sampling Network (AOSN) where moorings and/or buoys take the place of the support ship [21]. These "nodes" would act as communications relay stations, forwarding commands and AUV data via acoustic, radio, and satellite communications links. They would also allow the AUV(s) to recharge batteries, act as a navigational aid, and offer docks for vehicles to remain at while unused. Information sent to and from these nodes would link up to a global information network like the Internet to provide the researcher access to oceanographic sensors without ever leaving the office.

1.2 Our Approach

There are several problems facing remote control and monitoring of AUVs. Foremost is the difficulty of communicating acoustically in the ocean. While acoustic modem technology has made great advances in the last decade, such modems are still limited in bandwidth and reliability [18, 46]. The speed of sound in water incurs time delays, meaning an acoustic communications channel will never be as immediate as a tether. Complex acoustic propagation effects lead to blind spots, distortions, and echos which may be impossible to filter out, causing high error rates in data transmissions. On the land-based network, there are similar problems. Transient network glitches can cause additional time delays, as well as corrupting or destroying data packets. Connectivity with online oceanographic data archives, multiple users, as well as security also become issues. And finally, the lack of a support crew on-site means that the utmost care must be taken to assure the reliability of the system as a whole; the Phobos 1 spacecraft sent to Mars was lost due to a single errant command [25].

Research into the low-level aspects of these problems is left to those studying acoustic modems and network reliability. Instead, our research has investigated the effects of these problems on the higher level issue of communicating with an AUV. That is, given slow baud rates, large time delays, and unreliable and corrupting transmissions, how do you best monitor and control one or more AUVs collecting data in a remote location?

Figure 1-1 shows a logical development towards this problem. Other than a physical cable, the simplest communications link is a user connected directly to an AUV via a radio-frequency (RF) modem. At this level, the communications protocol and vehicle response to commands (both intended and erroneous) can be developed and evaluated. The next step extends the connection over a local area network (LAN) or the Internet. Issues of packet forwarding at the shore station, time delays, and additional sources of corruption must be addressed. Next, by replacing the RF modem with an acoustic modem, the reliability of the developed protocol can be tested in an underwater acoustic environment. The addition of nodes introduces another packet forwarding station. Finally, the problems associated with multiple vehicles and users can be addressed. Modems on multiple vehicles will interfere with each other, and so a communications system or management scheme must be used to allow the the vehicles to transmit data without conflicts.

Our research has focused on the first two of these steps. To gauge and address the problems of AUV communications in these situations, we have performed experiments with the Odyssey II AUV, first with a direct RF modem link to shore, and later by adding a connection to the Internet. We have explored data protocols, error correction, and vehicle control issues, and demonstrated remote launch, data visualization, and mission reconfiguration of AUV operations from a computer workstation located off-site. In addition, we have also investigated the multiple vehicle communications problem with a mathematical analysis of some common multi-node communications algorithms and how they apply to underwater acoustic communications.

1.3 Document Overview

This chapter has introduced the problem and described our general direction of research. Chapter Two will provide background material for our research. It will describe the Sea Grant AUVs, the underwater acoustic environment, and Internet transmission protocols

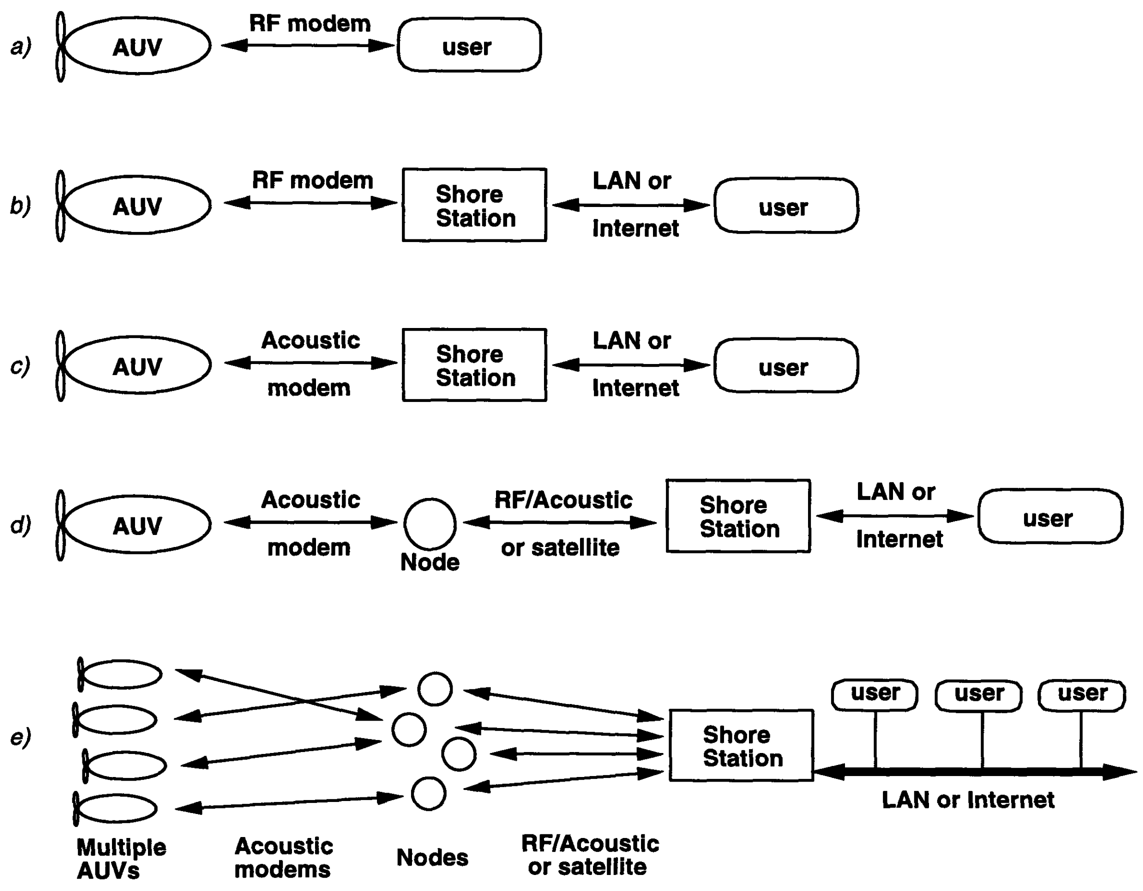


Figure 1-1: A logical progression of AUV control links. a) User in direction communication with AUV via a radio modem. b) Shore station in contact with an AUV via radio modem, user connection to the shore station by the Internet. c) Radio modem replaced by an acoustic modem. d) A relaying node placed between the AUV and shore station. e) Multiple AUVs with multiple nodes and multiple users.

in more detail. Chapter Three will describe the existing Odyssey II software and our additions to it. The original design criteria and rationale behind it, as well as an overview of our software implementation will be covered. Chapter Four will discuss our experiments and their initial results. Chapter Five will describe the issue of communications in multiple vehicle environments, and which solutions perform better under what circumstances. The final chapter will discuss the conclusions we arrived at from our work, as well as potential future areas of research.

Chapter 2

Background

2.1 Introduction

This chapter gives an introduction to the problem of remote monitoring and control of AUVs. The role of AUVs in the suite of oceanographic instrument platforms is presented along with a description of the MIT Sea Grant Odyssey II AUV. Problems facing underwater acoustic communications are described, along with current acoustic modem technology. Finally, networking protocols and Internet communications are introduced, and the issues of concern are discussed.

2.2 Autonomous Underwater Vehicles

AUVs play a role unfulfilled by the common dropped or towed oceanographic instruments. These instruments work well for large-scale surveys or investigation of a small site, but face problems with small-scale, high granularity surveys or deep-water operations. The cable tethering these instruments is expensive, in many cases equaling or exceeding the cost of the instrument platform. The cable is also subject to tangling and breakage, and imposes drag often making the platform dynamics unpredictable and difficult to control. These instruments are also limited in turn radius by the towing vessel, making a fine survey grid difficult or impossible to achieve. By virtue of their maneuverability, ROVs can make a fine survey grid possible. The presence of the tether, however, makes it difficult to perform surveys of large areas with an ROV.

Manned submersibles do away with the tether but must sustain a live crew with bulky life-support equipment. This seriously limits their endurance and mobility. Operations typically last less than 12 hours, most of which must be spent in descent and ascent. The additional equipment and the need for crew safety also drives up the cost; operating the Alvin costs from $21,000 to $28,000 per day [47]. In addition, the need to sustain a crew in a large pressure housing means rating the vehicle for greater depths involves military and/or civillian passenger craft certification, incurring even more expensive.

AUVs offer an unmanned platform without the limitations imposed by a tether. Because they do not need to sustain a crew, they can be made smaller than a manned vehicle. Without the need for a cable or life-support equipment, they are cheaper to manufacture and operate. Operating time, while not unlimited as with a tethered instrument, is typically greater than with a similar sized manned vehicle. However, because there is no tether and its associated drag, power consumption is less than with a ship and towed vehicle. The only real limitations of AUVs are their limited power supply, and their need for artificial intelligence sufficient for self-preservation.

2.2.1 Examples of AUVs

A variety of AUVs have been developed in recent years for military, scientific, and industrial purposes. Several notable vehicles are the Draper/DARPA UUV, the Advanced Unmanned Search System (AUSS), the Autonomous Benthic Explorer (ABE), and the Aqua Explorer 1000. The Draper/DARPA UUV, AUSS, and Aqua Explorer 1000 all possess dedicated, high bandwidth acoustic communication systems, and hence prior research with these vehicles is most relevant to the work in this thesis. The Autonomous Benthic Explorer (ABE) was designed with the capability for minimum acoustic command and control using simple coding of long baseline navigation system pings.

The Draper/DARPA UUV measures 36 ft in length with a 44 in hull diameter. The hull is watertight and is subdivided into partitions containing batteries, vehicle systems and electronics, and payload. Its original acoustic communications link, called the Adjustable Diversity Acoustic Telemetry System (ADATS) was typically capable of 200-400 baud at a range of approximately 1200 ft. This system was later upgraded to ADATS II, capable of 1200 baud at 3000 ft [41]. Most recently, joint research with the Woods Hole Oceanographic Institution produced an acoustic modem capable of approximately 5 kbps at a range of several kilometers in shallow water [27].

The Advanced Unmanned Search System was developed at the Naval Ocean Systems Center and measures about 4.3 m in length and 78 cm in diameter, weighing about 1 ton [50]. It was designed with a two-way acoustic link between vehicle and operator, and has the capability to receive acoustic supervisory commands, as well as acoustically transmit vehicle telemetry including digitized video from its cameras [49].

The Autonomous Benthic Explorer was developed at the Woods Hole Oceanographic Institution. It consists of an instrument platform attached below two floatation hulls to provide maximum stability. Length is about 2.2 m and displacement around 450 kg. ABE is designed for long-term deployment (up to a year) on the ocean bottom, where it initiates periodic bottom surveys as pre-programmed, or when receiving an acoustic signal from the surface. Its communications capability is limited to interrogate status, abort, and program select, but provides enough flexibility to avoid having to pre-program an entire 1-year mission in advance [53].

The Aqua Explorer 1000 was designed in Japan for inspection of underwater telecommunications cables. It is 2.3 m in length and 2.8 m in width by 0.7 m high, weighing about 500 kg. It uses both low-bit-rate and high-bit-rate acoustic links simultaneously, the former being used for control signals sent to the vehicle, and the latter for transmission of video signals from the vehicle [1].

2.2.2 The MIT Sea Grant Odyssey II AUV

MIT Sea Grant has pioneered the development of small AUVs for scientific exploration [8, 7]. The Odyssey AUVs are designed to provide a lightweight, flexible, affordable oceanographic survey platform which can be operated by a small crew using a minimum of support equipment. Instruments can be swapped in and out as they are needed, saving both weight and power.

The second-generation MIT Sea Grant Odyssey II AUV (Figure 2-1) is roughly 2.2 meters long and weighs about 120 kg. The outer hydrodynamic polyethylene fairing is flooded and carries instruments and batteries inside two 17 inch glass sphere pressure housings. The high density polyethylene (HDPE) has a specific gravity of 0.9, eliminating much of the need for ballast or buoyancy to counterweight the fairing. It is almost transparent to acoustic signals in the range of most oceanographic instruments, meaning acoustic transceivers can be mounted inside the fairing incurring no drag penalty. HDPE is also easy to work with

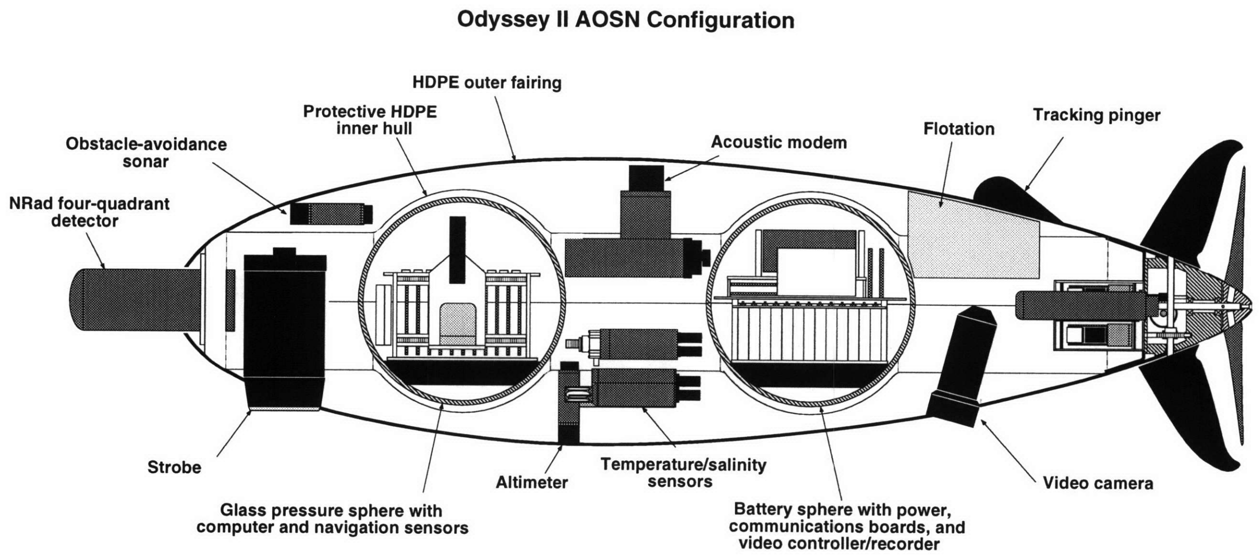

Odyssey II AOSN Configuration
HDPE outer fairing
Protective HDPE inner hull
Obstacle-avoidance sonar
NRad four-quadrant detector
Acoustic modem
Flotation
Tracking pinger
Strobe
Altimeter
Temperature/salinity sensors
Glass pressure sphere with computer and navigation sensors
Battery sphere with power, communications boards, and video controller/recorder
Video camera

Figure 2-1: Diagram of the MIT Sea Grant Odyssey IIb (courtesy Cliff Goudey).

and cheap to form, and offers enough resiliency to rebound from moderate impacts without substantial damage to itself or its contents [7].

The glass spheres are standard Benthos pressure housings used in other oceanographic instruments. The forward sphere contains the main vehicle computer and sensor electronics. The aft sphere contains batteries, thruster controllers, and additional sensor electronics. The two are connected to each other and instruments via pressure-rated wet-cabling mounted through penetrators. The spheres are held in place by another formed sheet of HDPE which provides additional stiffness and protection for the spheres. The spheres provide most of the buoyancy required for the vehicle to attain neutral-buoyancy, but additional lead weights or foam are used to trim the vehicle and compensate for the instrument packages currently installed.

Thrust comes from a custom-designed two-bladed fixed-pitched propeller powered by a Benthos thruster. The vehicle is maneuvered by four rear-mounted control surfaces (modified sailboard fins) linked to move as two units. The fins are controlled by actuators based on a design from the Woods Hole Oceanographic Institution. Power comes from 64 silver-zinc batteries mounted in the aft sphere which provide approximately 1.1 kW-hr of energy and weigh 8 kg, enough to provide 6-10 hours of use at moderate speeds. Top vehicle speed is approximately 5 knots, while the lower speed limit is defined by a loss of maneuverability at around 0.5 m/s (0.25 knots).

The main computer is a 40 MHz Motorola 68030 CPU with coprocessor mounted on a

VME-bus system supplemented by 8 MB of RAM and a 120 MB hard disk. Interfaces to onboard instruments and to off-vehicle computers are provided by ethernet and serial ports. The computer runs the OS-9 realtime multitasking operating system.

The Odyssey software suite is a constellation of applications which are used to prepare, test, run, and analyze missions [9]. These include: mission configuration software, vehicle control code, data parsing routines, mission data analysis/visualization, simulation models, and data structures. The high-level control software is a behavior based layered control architecture, which allows relatively complex behaviors (e.g. a vertical yo-yo concurrent with a horizontal survey) to be easily produced [5, 9]. The vehicle software has been proven reliable in extensive field operations with Odyssey II AUVs over the past several years, including operations in under-ice, deep-ocean, and high-current environments. The Odyssey software is described in more detail in Chapter 3.

2.3 Underwater Acoustic Communications

Acoustic communication in the ocean is a challenging research problem [24]. A multitude of factors not often encountered in other communications channels, such as substantial multipath propagation, corrupt and distort sound as it travels from source to receiver. The long signal travel times involved make it impossible for the vehicle to instantly respond to commands. In addition, retransmission is an impractical means of error correction, and redundancy and error correction coding must be added to the original signal [15]. In a multi-vehicle and/or multi-instrument environment, several modems may be competing for the same frequency space, and a procedure for allocating acoustic bandwidth must be in place.

Despite these difficulties, advances in digital signal processing, acoustic channel modeling, and communications theory have led to much improved performance of underwater acoustic modems. At a range of around ten of kilometers, transmission rates of 1 to 10 kb/s have been achieved with acoustic modem technology [17].

2.3.1 The ocean as an acoustic channel

Unlike air and outer space, water acts as a low-pass filter towards electromagnetic (EM) radiation, absorbing most frequencies within a few meters or less. The extremely low frequency radio waves which do penetrate are used for military communication with submarines, but they require very large antennas and have very slow transmission rates. This characteristic of water precludes the use of EM signals for underwater communications for most applications, including communication with AUVs over 1-10 km distances. Communications must instead rely on the transmission of sound. While there are few problems with direct-line, short range communications (several hundred meters or less), a number of complications arise at greater distances.

Sound spreads and is absorbed by various oceanic processes as it travels. Geological, biological, and man-made noises (most notably, noise from the AUV itself) are always present and can mask incoming acoustic signals. Turbulence and other variations in temperature, density, and water composition cause fluctuations in the local sound speed. The presence of a surface and bottom and a gradual change in the sound speed profile with depth gives rise to multipath. Finally, sound travels at a relatively slow 1500 m/s, which results in round-trip signal travel times of approximately 1.3 seconds for every kilometer separating the source and receiver.

The dissipation of sound energy due to spherical spreading and absorption can be simply characterized as a transmission loss, H. The logarithmic sound pressure level at a point, L_p, can be given as a combination of the sound source level, L_s, and the transmission loss.

$$L_p = L_s - H \tag{2.1}$$

If the distance separating the source and measuring point of L_p is r, then H can be given as

$$H = 20 \log \frac{r}{r_{\text{ref}}} + \alpha r \tag{2.2}$$

where the first term represents spherical spreading, and the αr term represents absorption due to viscosity and relaxation processes associated with the structure of water and various dissolved salts [23]. Taking these processes into account, α can be determined by [23]

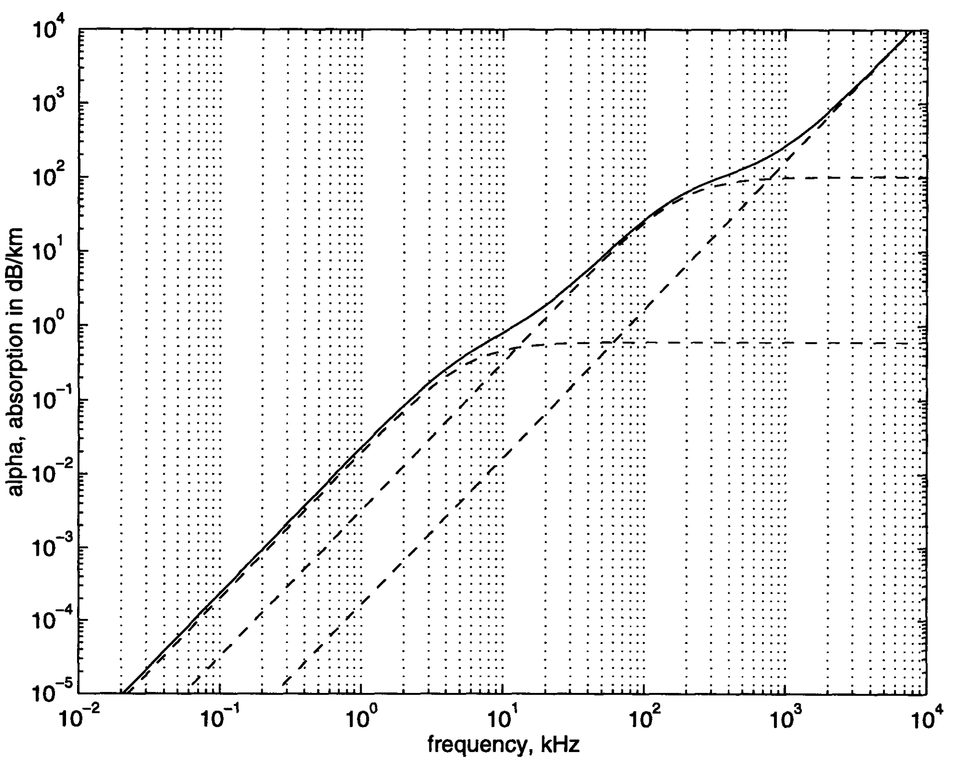


Figure 2-2: Absorption of sound in seawater due to various processes (from [23]).

$$\alpha = \alpha_2 + \alpha_3 + \alpha_4$$

$$\begin{aligned}
\alpha_2 &= 2.94 \times 10^{-2} f_3^{-1} f^2 \\
\alpha_3 &= 2.03 \times 10^{-2} S(1 - 6.54 \times 10^{-4} P) \frac{f_3 (f/f_3)^2}{1 + (f/f_3)^2} \\
\alpha_4 &= 1.10 \times 10^{-1} \frac{f_4 (f/f_4)^2}{1 + (f/f_4)^2} \\
f_3 &= 1.55 \times 10^4 T \exp(-3052/T) \\
f_4 &= 1.32 T \exp(-1700/T)
\end{aligned}$$

where f is the transmitted sound frequency in kHz, and S, P, and T are the salinity in ppt, pressure in atm, and temperature in $^\circ K$. This yields α for r given in dB/km. Figure 2-2 gives average α values for an open ocean environment over an average depth of 6 km and matches well with measurements made at sea.

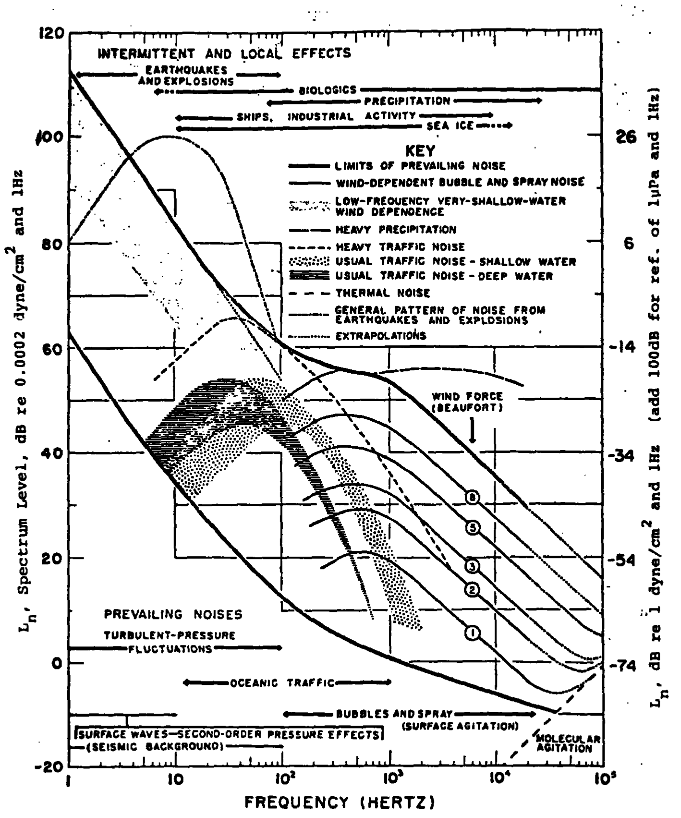


Figure 2-3: Noise levels in the open ocean at various frequencies (from [52]).

Absorption increases at higher frequencies and often imposes an upper limit to the frequency chosen for an application. The presence of noise in the ocean often provides a lower bound. Figure 2-3 shows noise levels for a typical ocean environment. Minimizing the transmission losses due to these two factors, we obtain the frequencies commonly used in underwater acoustic modems (and other applications for that matter) of 10-35 kHz.

The speed of sound is not uniform throughout the ocean, but is commonly stratified by depth. This gives rise to the phenomenon of refractive multipath. Sound traveling from a low sound-speed layer into a high sound-speed layer is bent back towards the low-speed region. In addition, sound can be reflected off the ocean surface and bottom. The magnitude of these reflections is dependent on the signal's arriving angle and frequency, as well as bottom composition. This means sound spreading radially from a source does not propagate uniformly. At any given destination point, sound rays from multiple departure angles can meet, or be completely absent. Where multiple rays meet, they have taken completely different paths and are almost always out of synchronization and phase. A considerable amount of acoustics research has addressed the problem of extracting the original version of a signal that has become spread out into multiple signals and faded due to phase interference between these duplicate signals. On the other hand, the sound-speed profile can contrive to create a region where no rays from a source can arrive in certain regions. These shadow zones represent areas where communication signals cannot reach and vehicle control would be lost. Because of the asymmetrical nature of multipath, vehicle transmissions may still be heard while the AUV is in a shadow zone, or the opposite situation may arise, where the AUV cannot be detected but can still receive commands.

Due to the slow speed of sound through water, traditional forms of error correction become inefficient. In high-speed data networks, the most common error-correction scheme is to retransmit the data. With the lengthy transmission delays in an acoustic underwater environment, retransmission as an error correction scheme can take a significant amount of time.

If the probability of an error is P_{err}, we can find the average number of times a message must be retransmitted until it is received correctly. First we have all messages which are correctly received on the first transmission, plus the messages which must be re-transmitted, plus messages which must be re-re-transmitted, and so on. On average, the number of retransmissions will be

$$(P_{err} + P_{err}P_{err} + P_{err}P_{err}P_{err} + \cdots) = \sum_{n=1}^{\infty} P_{err}^{n} \tag{2.3}$$

Since for $x < 1$

$$\sum_{n=0}^{\infty} x^{n} = \frac{1}{1-x} \tag{2.4}$$

we have

$$\sum_{n=1}^{\infty} P_{err}^{n} = \frac{1}{1-P_{err}} - 1 \tag{2.5}$$

A retransmit cycle requires three times as much time as an errorless transmit: the original leg from source to receiver, the retransmit request from receiver to source, and the retransmission from source to receiver. Add to this the average number of messages that get through on the first try, P_{noerr}, and substitute P_{noerr} for $1 - P_{err}$. Multiply by the one-leg transmission time, t, to get the average error-prone transmission time if we use

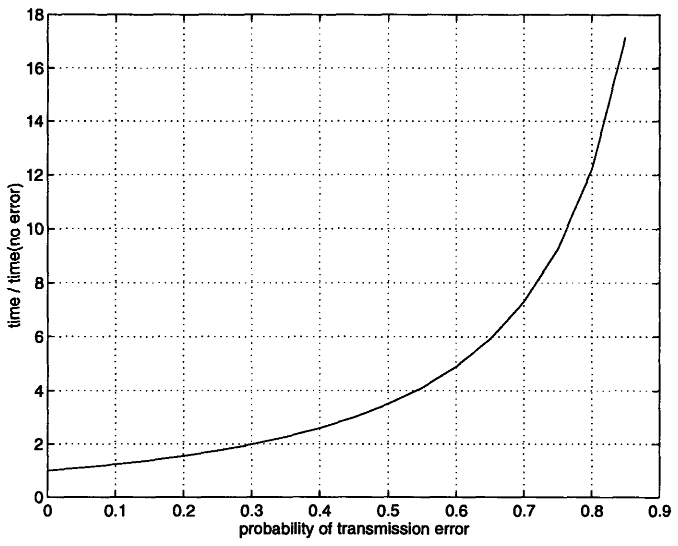


Figure 2-4: Mean error-prone transmission time using retransmission as a means of error correction, relative to errorless transmission time.

retransmission as a means of error correction.

$$t_{avg} = [P_{noerr} + 3(\frac{1}{P_{noerr}} - 1)]t \tag{2.6}$$

or

$$t_{avg} = [P_{noerr} + \frac{3P_{err}}{P_{noerr}}]t \tag{2.7}$$

The results of this equation are plotted in Figure 2-4 relative to the time for an errorless transmission, and in Figure 2-5 as a function of range and P_{err}.

Finally, if there is any relative motion between the source and receiver, the signal can become doppler shifted. This usually results in a reduction in the usable frequency bandwidth, as the modem hardware must compensate to eliminate any uncertainty about the frequency of the detected signal. Strategies for accomplishing this are described in the next section.

2.3.2 Acoustic modem technology

Acoustic modems fall into two categories - incoherent modulation systems such as frequency shift keying (FSK), which do not rely on the phase of the received signal; and coherent modulation systems such as phase shift keying (PSK) and quadrature amplitude modulation (QAM), which use phase information. Because multipath typically spreads the arrival time

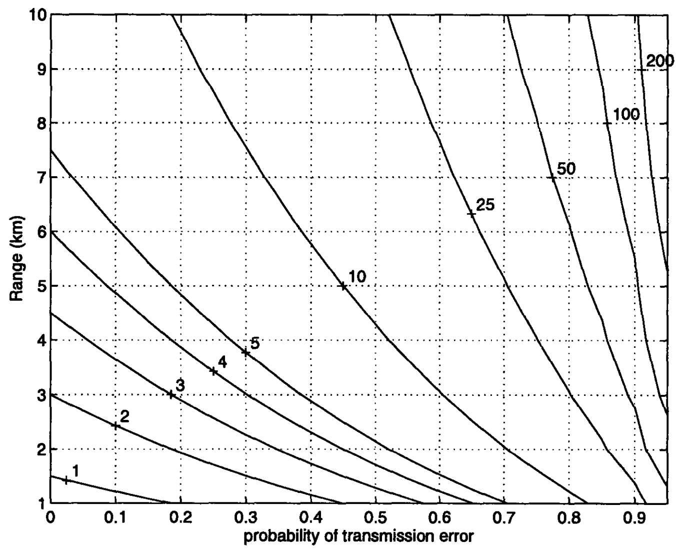


Figure 2-5: Mean error-prone transmission time for a variety of ranges and error rates, in seconds.

of a signal over several milliseconds, coherent systems work best in low-multipath, stable-phase environments such as vertical channels and short ranges; incoherent systems are better suited for high-multipath environments such as shallow water and moderate ranges of 1-10 km. However, incoherent systems usually cannot match the bandwidth of coherent systems because the possibility of inter-signal interference due to multipath spreading limits the rate at which a frequency can be modulated [35].

Since retransmission is usually not a viable means of error correction, some form of forward error correction coding must be used unless errors are acceptable. The performance for several codes varies depending on environmental circumstances [17]. Generally, efficient use of bandwidth requires greater coding complexity. Since error coding can be implemented in the modem or on the main computer prior to transmission, this is a potential area for bandwidth optimization by software adapatation based on the mission profile and acoustic situation encountered.

Doppler shifts characterize themselves as uncertainties in the frequency of the arriving signal. In the 30 kHz range, a 1 m/s doppler shift results in a frequency shift of 30 Hz. Since the source and receiver can be moving both towards or away from each other, the total uncertainty is actually double this. To compensate, most incoherent systems space their transmit frequencies far enough apart that doppler shift is not a problem. Other approaches include coding information that allows detection of doppler shift, and tracking

Table 2.1: Recent acoustic modem systems.

Developer	Data rate	Range	Modulation	Reference
Datasonics	5 kbps	1 km	incoherent	[16]
Datasonics	1200 bps	10 km	incoherent	[22]
WHOI	5 kbps	5 km	incoherent	[31, 32]
WHOI	5 kbps	shallow, under ice	coherent	[27]
JAMSTEC	16 kbps	6.5 km vertical	coherent	[36]
IFREMER/ORCA	19.2 kbps	2 km vertical	coherent	[3]
Oki Elec.	500 kbps	60 m	coherent	[29]

the source and receiver so as to predict the doppler effect. The former reduces the available signal bandwidth, while the latter may be impossible if both the source and receiver are mobile [46].

Table 2.1 lists some acoustic modem systems which have been developed. For shallow-water AUV operations in the 1-10 km range, data rates of 1-5 kbps can be expected in optimal conditions [17, 46]. For deep water missions where a vertical channel may be available, higher data rates are achievable.

2.4 Networked Communications

2.4.1 The Internet as a communications channel

The Internet was designed to provide reliable high speed and wide bandwidth communications, and compared to underwater acoustic communications imposes few restrictions on communications. Network programming standards such as Sockets allow virtually unchanged source code to work on nearly all types of UNIX computer systems [45]. A potential difficulty arises from the Internet using the Transmission Control Protocol (the TCP in TCP/IP) - a point-to-point protocol - as its transport level protocol. However, there are alternatives available which, while limiting the networking options available, do provide a solution to the design faults of TCP.

Internet Protocols

The Internet is not a single network but rather a conglomeration of thousands of independent local area networks (LANs) running a diverse variety of network hardware and protocols. Under most circumstances, it is very difficult or impossible to interconnect two

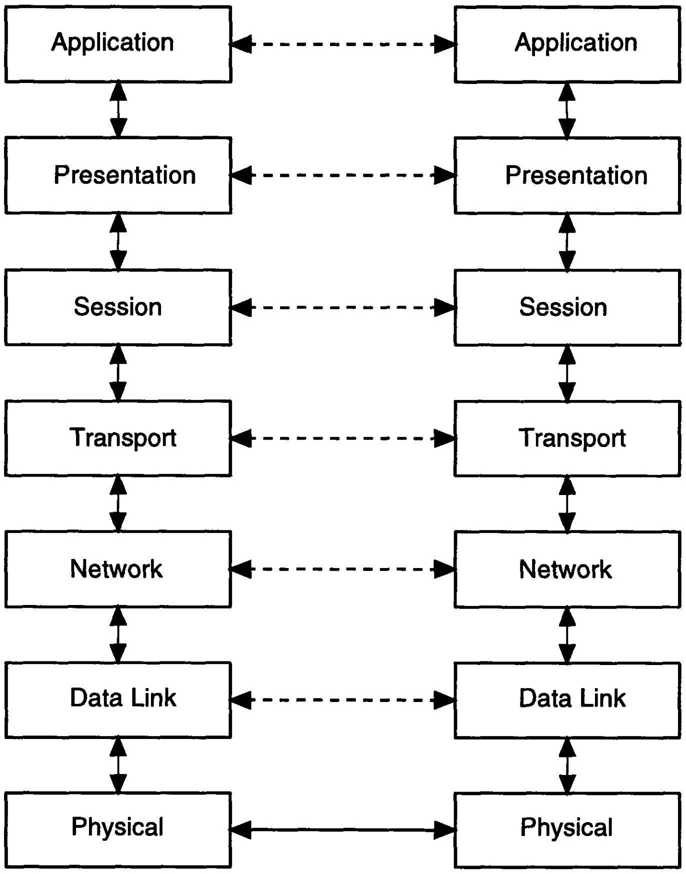


Figure 2-6: Open Systems Interconnection network protocol hierarchy model (from [45]).

different networks. The key to joining all these networks together to form the Internet lies in encapsulating and layering. This allows the network to appear as a simple direct connection to the linked computers and data, hiding the specifics of the underlying protocols and hardware. The commonly used open systems interconnection model (OSI) consists of seven layers for describing any application communicating over a network, shown in Figure 2-6 [45].

At the lowest level is the hardware — whether data is sent as electrical on-off impulses, modulated electrical frequencies, or flashes of light. Next is the hardware data link which determines what these flashes or impulses mean in terms of bits. Above this is the network protocol, which determines the type of network (e.g. Novell, Internet, etc.). Above the network layer is the transport layer which determines how data sent over the network is broken up and reassembled.

On the Internet, the network protocol is the Internet Protocol (IP), which handles such tasks as breaking data into packets and routing them between networks around the world. Above the IP layer is the Transmission Control Protocol (TCP) which provides such services as opening and maintaining a connection, and reordering packets which may arrive out of order at the destination. Above this layer are various application protocols such as telnet and ftp, which provide their own services such as a common format for creating and receiving transmitted files. And finally there is the data which is sent over the network.

As data progresses through this hierarchy, each layer need not be concerned with what layers come before or after it. A layer just treats the data protocol headers from the next layer up as any other data stream and encapsulates it within its own headers. This continues on through the entire chain until the hardware is finally able to transmit what it sees as a simple data stream. When the data arrives at the final destination, it goes through the reverse process, each layer stripping away its header until the original data is revealed.

The protocols that define the Internet are the TCP and IP layers (often referred to as TCP/IP). Nearly all the services and applications commonly used on the Internet, such as electronic mail, telnet sessions, the world wide web, and file transfer, use TCP/IP. If all that is required is to send a stream of data from one computer to another, TCP/IP (or any of the services built on top of TCP/IP such as telnet or ftp) works fine. However, if more complex communications is desired, such as sending data from one computer to several others, a problem arises because TCP/IP is a point-to-point protocol.

Point-to-point protocols were developed with only communications between two machines in mind, and are wasteful of network resources if many machines request the same information. One machine establishes a connection with the other, and the two transfer data. If a third machine wants the same data, it must open another connection and the first machine must retransmit it. As more machines ask the first machine for the data, the network bandwidth becomes saturated transmitting the same thing over and over. This is normally not a problem if only a few connections or small amounts of data are required, but otherwise the level of network traffic can quickly exceed the available bandwidth. For example, in the Haro Strait experiment [44], the link to the Internet was nothing more than a modem and telephone line. Had many researchers around the world wanted to view the raw data as it came in, the bandwidth would have quickly been saturated. The problem is even worse if several machines must all share data between each other, as the case may be if distributed computing resources are used to analyze the incoming data. The required network bandwidth then goes from scaling linearly to polynomially with the number of machines.

Since the problem is inherent in the protocol that forms the basis for the Internet, there is no simple way around it. Point-to-multipoint protocols such as the User Datagram Protocol (UDP) exist, but their use precludes the use of TCP, meaning they will not normally propagate over the Internet. UDP will propagate over a subnet, but Internet routers normally ignore these packets. Fortunately, this is not a new problem, and the Multicast Backbone, or MBone, was created to address it [37]. The MBone uses special routers available on a portion of the Internet to "tunnel" UDP packets over the Internet's data lines. Data which is multicast is sent over the entire MBone, but only once. All machines interested in the data can listen in on the single transmission rather than the originating machine having to send a separate copy to each machine.

A potentially useful multicast standard is Distributed Interactive Simulation (DIS), which has been developed to allow multiple wargame simulations to interact with each other in the same virtual environment [14]. DIS defines a common set of datagram packet formats which independent simulators can use to communicate with each other. These packets include a unit's identity, position, velocity, projectiles fired, etc. To further reduce network bandwidth, dead-reckoning is used — every participating simulator uses the received position and velocity of units it is not simulating to dead-reckon their position. When a machine simulating a unit detect that the simulated position (or actual position in the case of a real vehicle) differs from the dead-reckoned position by a predetermined about, only then is a new packet sent out.

While initially we considered making our system DIS-compatible, we decided that the extraneous data fields in a DIS packet (e.g. logistics data units) were unnecessary for these initial experiments. We chose instead to develop our own packets which contained only the information we needed, but were easily reconfigurable. However, because there is a large amount of DIS source code freely available, we believe DIS may be a useful standard to investigate in the future.

2.5 Summary

This chapter has introduced AUVs and their role in oceanography. Problems with underwater acoustic communications, networking protocols, and Internet communications were discussed. The next chapter will describe our modifications to the MIT Sea Grant Odyssey II AUV software to account for these problems and incorporate the networking technologies.

Chapter 3

Software Design

3.1 Introduction

This chapter describes the Odyssey II software developed at the MIT Sea Grant AUV Lab. It begins with an overview of the Odyssey II software design and philosophy, detailing the software's control and data structure already in place. Differences in a simulated mission, real mission, and real mission with communications are highlighted, along with changes made to the vehicle software in the course of this thesis to allow communications. The shore station relays messages between the user and vehicle, and its software is described. Finally, the data display and control interfaces running on the user's computer are described.

3.2 Vehicle Software Design

The Odyssey control software was designed and implemented by Bellingham and Leonard [4, 5, 9] with two major goals in mind — ease of use and reconfigurability. The vehicle consists of many separate subsystems, most of which require a considerable amount of programming in order to function properly. Fin actuators must be controlled according to a well-tested dynamic model in order to move the vehicle around properly. Navigation signals received must be processed to filter out spurious signals and accurately fix the vehicle's location according to a variety of different navigation schemes. It would be an exercise in frustration for the vehicle operator to attempt to control the vehicle based only on raw inputs and outputs with the sensors and actuators. To address this problem, the Odyssey software implements behavior based layered control [9].

Since the vehicle is designed to be small, and instrument platforms swapped in and out as needed, the software must be easy to reconfigure. To achieve this, a central data structure (sometimes called a blackboard) is used. The vehicle's data structure contains all the variables used by the sensors and actuators during a mission. When a new instrument or modified actuator is installed, the data structure must be modified or lengthened to reflect this change. Since the data structure is key not only to vehicle operation, but also to mission configuration and later data analysis, these changes tend to propagate throughout the vehicle code and can result in these dependent functions failing in unpredictable ways. To solve this dilemma, the Odyssey II uses a name-referenced data structure. The code is written so as to expect a data structure of arbitrary length. Each component of the data structure is indexed, and the name of each variable is assigned an index number. In this way all changes can be propagated throughout the code by simply recompiling.

3.2.1 Layered control

Layered control represents a departure from many of the high-level artificial intelligence (AI) schemes developed to control robots. In most AI systems, the software attempts to analyze and comprehend the environment and situation the robot is currently in, then selects the most appropriate action for achieving the desired goals. The primary faults of these systems are that data about the environment is rarely complete and accurate, and even with comprehensive data, fully analyzing the situation is computationally intensive and a difficult programming task [39]. In layered control, rather than attempting to understand the environment and develop a response, the software consists of many simple, individual responses to various commands and stimuli in the environment. Each of these responses form a behavior, all or some of which may be active at any time. When behaviors conflict, an arbitration scheme determines which behavior should be given priority. In this way, many complex robot interactions with the environment can be created using just a few simple-to-program behaviors [12, 13].

For example, suppose one wished to program a robot to follow a zig-zagging wall. In traditional AI, the robot would use its sensors to determine where the wall is and build a map of its immediate vicinity. It would then analyze this map to determine a likely continuation of the wall and move in that direction. When a new section of wall was encountered, it would repeat this process. Resources must be devoted to build and maintain this map,

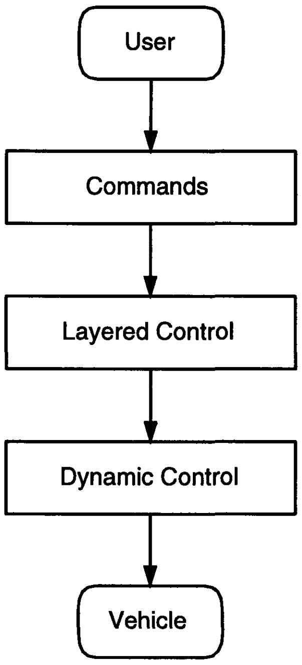


Figure 3-1: Levels of control in vehicle software

which may become inaccurate through simple sensor failures.

With layered control, no real attempt need be made to create an internal map. The robot could consist of several simple behaviors. One could prevent it from hitting a wall (maintain a minimum distance from all known walls). Another behavior, operating at a lower priority, could keep the wall off to the side as the robot moved forward. Yet another behavior could check for an infinite circling pattern, and direct the robot to continue straight in a random direction for a time to get away from the obstacle. In this way, by merging several simple behaviors, the complex behavior of following a wall emerges.

The Odyssey II software uses a three-tiered layered control architecture, of which two levels are fully implemented [9]. These levels are shown in Figure 3-1. At the top level, which has not been fully implemented on the Odyssey II, are user-generated commands. These commands, along with sensor information, are passed on to layered control, which consists of various behaviors. Outputs from these behaviors are combined to generate a single set of commands sent to the lowest level, dynamic control. The dynamic control level communicates directly to the vehicle hardware — moving fins, varying propeller speed, and turning equipment on and off.

The command level on the Odyssey II currently consists of specifying the layered control behaviors and their priorities for each mission. A mission is created by varying the priority

```
state: nrad survey test mission            behavior: mission_timer 1
 sensor:          u_debug_var(int) 1        b_arg:                      time(s)  900
 sensor:     c_modem_active(bool) 1        behavior: set_rudder 2
 sensor:      c_weight1_drop(bool) 0        b_arg:                 rudder(deg)  20.0
 sensor:      c_weight2_drop(bool) 0        b_arg:                    depth(m)  2.0
 sensor:             real_wdw1(N) 0         b_arg:                 speed(m/s)  0.6
 sensor:             real_wdw2(N) 0         b_arg:                      time(s)  900
 sensor:            real_veh_w(N) 1173     behavior: survey_dead_reckon 3
 sensor:            real_veh_b(N) 1173      b_arg:        trans_heading(rad)  0
 sensor:          uo_control_kpp 1          b_arg:              trans_time(s)  30
 sensor:          uo_control_kdp 5          b_arg:        heading_leg1(rad)  0
 sensor:           uo_control_kip 0         b_arg:               time_leg1(s)  60
 sensor: uo_control_pitch_lim(deg) 30       b_arg:        heading_leg2(rad)  1.57
 sensor: uo_control_depth_lim(deg) 0.3      b_arg:               time_leg2(s)  15
 sensor:          uo_control_kph 1          b_arg:                  cycles(#)  4
 sensor:          uo_control_kdh 1          b_arg:                   depth(m)  2.0
 sensor:           uo_control_kih 0         b_arg:      transit_speed(m/s)  1.05
                                            b_arg:       survey_speed(m/s)  1.05
```

Figure 3-2: A sample Odyssey II mission file. On the left, various flags, sensors, and actuators are activated and initialized. On the right is the actual mission, based on behaviors in the vehicle's behavior library. In this example, the vehicle would start a dead reckoned survey for 4 cycles, then enter a circling pattern. After 900 seconds, the mission times out and the vehicle shuts down and surfaces.

and timeouts for each behavior. When the time on a certain behavior expires, the vehicle moves on to the next behavior on the command stack. This list of behaviors and priorities is placed in a Mission file which is read into the command stack by the vehicle computer at the start of the mission. Figure 3-2 shows a typical Odyssey II mission file.

Ideally, the command level would be an abstraction of vehicle actions, such as collecting data from a specified location with a specified resolution. The command level would then convert this command into suitable layered control behaviors to get the vehicle to the site, collect the data, and return home.

The middle level is layered control, consisting of the behaviors themselves. These behaviors can receive sensor inputs and generate commands sent to (or through) the dynamic control layer. Behaviors range from simple to complex. For example, in the mission shown in Figure 3-2, `mission_timer` shuts the vehicle down after a user-selectable amount of time, 900 seconds in this case, ensuring that a mission will end even if an infinite loop has accidentally been programmed in. The behavior `set_rudder` sets the rudder at 20 degrees, putting the vehicle in a tight circling pattern. On the other hand, `survey_dead_reckon` is a complex behavior which moves the vehicle away from its launch point in a specified direction, then does a lawnmower-type rectangular survey using dead reckoning for navigation. It is composed of several steps, including calling other behaviors in the course of its operation.

Most layered control implementations use some sort of arbitration scheme to merge conflicting commands from multiple behaviors into one output command. The Odyssey II uses a simpler implementation in which a behavior passes its command down to the next higher priority behavior, which passes its command down, and so on.

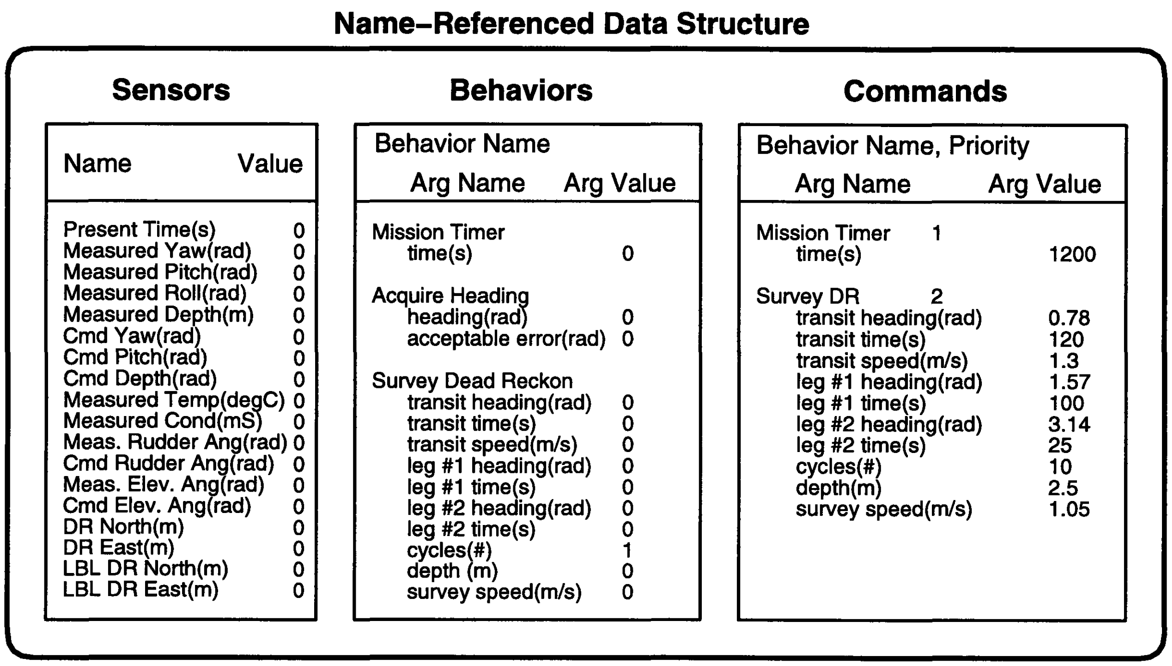
At the lowest level is dynamic control, which converts simple instructions such as heading, pitch, and depth into actuator commands required to fulfill these instructions. In this way, control laws governing the fins and thruster are handled separately from higher behaviors, and programmers of these behaviors need not concern themselves with actuator operation. This level can, of course, be bypassed if needed, as is the case when developing these control laws.

3.2.2 Vehicle data structure

The Odyssey II is designed to be a flexible, reconfigurable instrument platform. Carrying all instruments and sensors designed for use aboard the vehicle is impractical. In addition, packages for the vehicle are being developed at several companies around the country. The vehicle software must be robust and flexible enough to accommodate all these different configurations without requiring a complete rewrite of the code every time a new package is added [9].

The Odyssey II data structure has three components — sensor data, behaviors, and the command stack (Figure 3-3). Sensor data are all the variables used by the sensors and actuators during a mission. Examples include the time, yaw, pitch, roll, fin measured and commanded positions, and universal coordinates based on dead reckoning and the long baseline (LBL) navigation array. Each sensor data type has a name (an index) and a value as well as several flags. Behaviors begin with the name of the behavior and contain more sensor data — variables used within the operation of the behavior. Values for sensor data and behavior sensor data are set to the variable's default value. The command stack is made of a list of behavior names, their priority, and sensor data name and initial value. In completely autonomous operations, the command stack is static throughout a mission, having been specified in the mission file.

This data structure is central to the vehicle's operation. The software aboard the vehicle runs in a 5 Hz cycle, executing a loop every 0.2 sec. This loop begins with data input and


Name-Referenced Data Structure

Sensors

Name	Value
Present Time(s)	0
Measured Yaw(rad)	0
Measured Pitch(rad)	0
Measured Roll(rad)	0
Measured Depth(m)	0
Cmd Yaw(rad)	0
Cmd Pitch(rad)	0
Cmd Depth(rad)	0
Measured Temp(degC)	0
Measured Cond(mS)	0
Meas. Rudder Ang(rad)	0
Cmd Rudder Ang(rad)	0
Meas. Elev. Ang(rad)	0
Cmd Elev. Ang(rad)	0
DR North(m)	0
DR East(m)	0
LBL DR North(m)	0
LBL DR East(m)	0

Behaviors

Behavior Name		
	Arg Name	Arg Value
Mission Timer		
	time(s)	0
Acquire Heading		
	heading(rad)	0
	acceptable error(rad)	0
Survey Dead Reckon		
	transit heading(rad)	0
	transit time(s)	0
	transit speed(m/s)	0
	leg #1 heading(rad)	0
	leg #1 time(s)	0
	leg #2 heading(rad)	0
	leg #2 time(s)	0
	cycles(#)	1
	depth (m)	0
	survey speed(m/s)	0

Commands

Behavior Name, Priority		
	Arg Name	Arg Value
Mission Timer 1		
	time(s)	1200
Survey DR 2		
	transit heading(rad)	0.78
	transit time(s)	120
	transit speed(m/s)	1.3
	leg #1 heading(rad)	1.57
	leg #1 time(s)	100
	leg #2 heading(rad)	3.14
	leg #2 time(s)	25
	cycles(#)	10
	depth(m)	2.5
	survey speed(m/s)	1.05

Figure 3-3: A subset of the name-referenced data structure of the Odyssey II software

output as shown in Figure 3-4. Data from the instrument platforms and actuators are read into their appropriate variables in the data structure. In the next step, this data is processed. e.g. time-of-reception values from the LBL transducer are converted into navigation position. This step generates more data saved as additional variables in the data structure. Next the behaviors currently in effect use the current data to determine what the vehicle should do next. Their commands are saved in the data structure, where the dynamic control routine converts them into appropriate actuator signals. Finally, the data structure is logged to disk. To reduce disk space usage, only variables which have changed are logged.

In addition to being the heart of the vehicle's operational software, the data structure is central to pre- and post-operation routines. The Mission file which will be read into the command stack must be based on the data structure compiled into the code. The `parse` routine, which processes data logged during a mission, must know what the data structure is in order to extract the variables the user wishes to plot or process further. This information is contained in a header in the data file. The simple act of adding a new sensor data type to the data structure has far-reaching effects throughout all the code used with the vehicle.

By using a name-referenced data structure, all the code can be written to expect changes in the data structure. Rather than making the data structure a fixed size, it is coded as

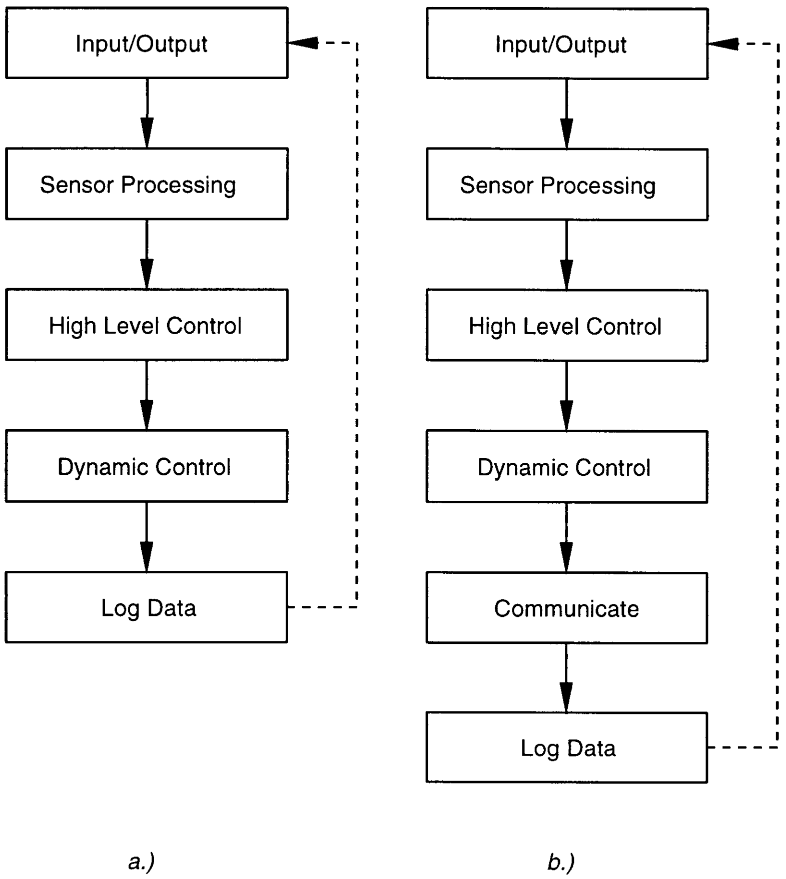


Figure 3-4: Internal processing loop of the Odyssey II software. a) In autonomous operations. b) With communications.

an indexed array whose length will be determined at compile time. Each variable is given an index number which is assigned to that variable's name. When adding new equipment to the vehicle, any new sensor variables are added to the code. Sections of code that can safely ignore the new sensor variables can look up the variables they need via the name, even though the actual position in the data structure may have changed. Routines that must handle the entire data structure (e.g. logging) integrate the new variables upon recompilation. The same procedure applies to new or modified behaviors. In this way, equipment can be added and vehicle configuration changed with no or minimal modifications to the base vehicle code.

3.3 Design Criteria and Issues with Communication

In completely autonomous operations, the MIT Sea Grant Odyssey II AUV is programmed with a complete mission prior to launch (Figure 3-5b). The vehicle is hooked up to a shore computer via ethernet wet cabling, and the mission configuration files are written or modified. An init file, which tells the vehicle software which instruments to activate during the mission, is also created. During launch, the vehicle runs a countdown which defaults to 60 seconds. The ethernet cable is disconnected, and the vehicle is manually submerged. At the end of the countdown, the mission begins and the vehicle departs under its own power. At the end of the mission, the vehicle powers itself to the surface and ends the mission, cutting off power to the fin actuators and propeller. The vehicle is retrieved and reconnected to the shore computer, and the data logfile of the mission is downloaded to the shore computer. This logfile is parsed with a program called `parse` to convert the variables the user wishes to view into a Matlab-readable form. The program `parse` also generates a Matlab script, called `review`, which displays the graphs and plots most commonly used. The user then runs `review` to view the data from the logfile.

The Odyssey II software is also designed to run on Unix workstations as a simulator — running the original software but without the vehicle hardware. As shown in Figure 3-5a, in the simulator, there is no need to modify an init file, launch or recover the vehicle, or download the data.

Remotely controlled operations required or allowed several changes to this procedure. First, in the vehicle software, a communicate routine was added immediately after the dynamic control routine (Figure 3-4). This communicate routine selects and sends data from the data structure to the modem for transmission to shore. It also receives new commands and inserts them into the data structure. For actual vehicle operations, a communication routine running as a separate process was used instead of inserting the routine directly into the vehicle code (Figure 3-5c). This ability to exchange data and commands before completion of the mission creates a new flowchart for vehicle operations. Whereas previously the user would sit and wait between the AUV launch and recovery phases, the user could now receive data as the vehicle collected it, and interactively modify the mission in progress.

Because of bandwidth limitations, the data sent back to the user contains only a portion of the entire data logfile. For our software implementation, we had to decide which

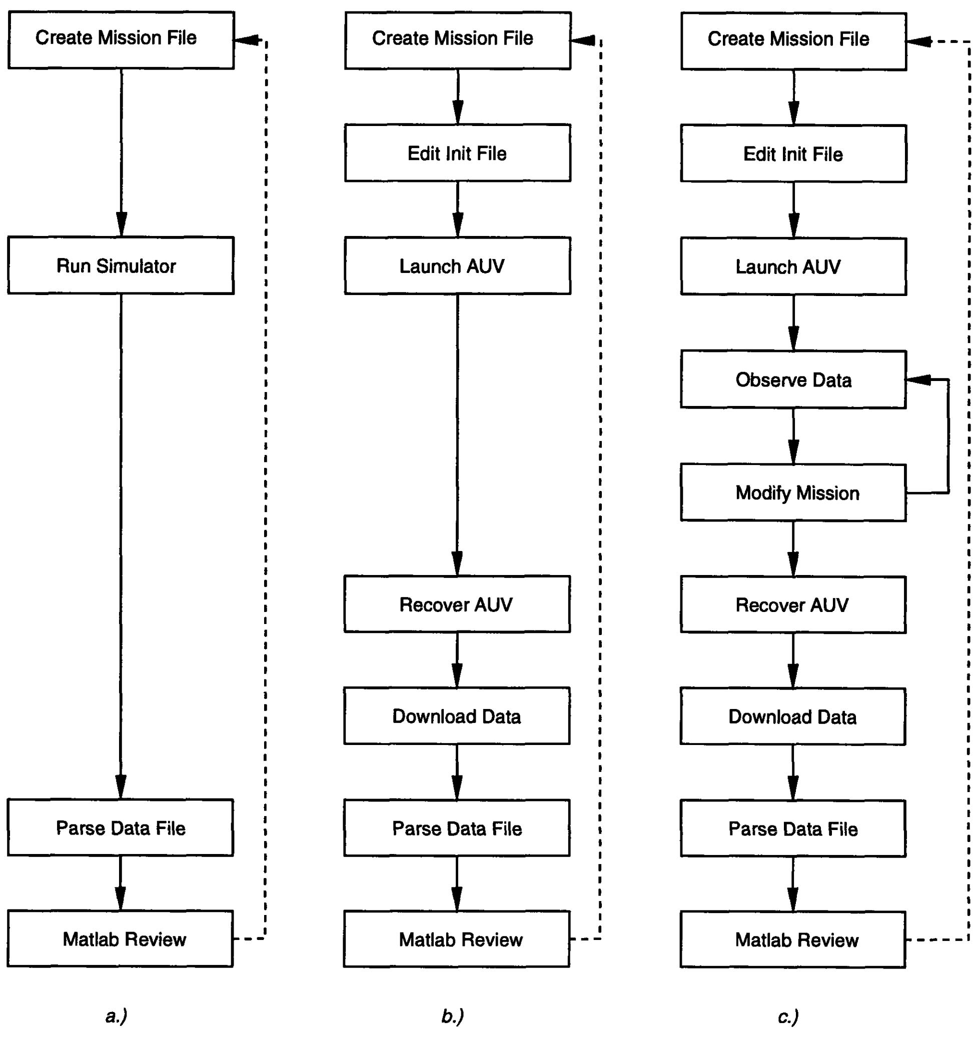


Figure 3-5: Flowchart for Odyssey II missions. a) Simulated mission. b) Real mission. c) Real mission with communications.

variables to send, what format to send it in, and implement error checking and possibly error correction. Also, because the `parse` and Matlab `review` routines were designed to work with unchanging data, new routines were developed which could accommodate data continuously streaming in. The possibility of sending vehicle data over a network led to another question: What if more than one person wished to observe the data as it came in? Because of the bandwidth problems of multi-node Internet communications described earlier, our software utilized the MBone to multicast data so all interested parties could observe it.

The ability to modify the mission file also presented challenges. Because of bandwidth limitations, uploading an entirely new mission file, while possible, was considered unnec-

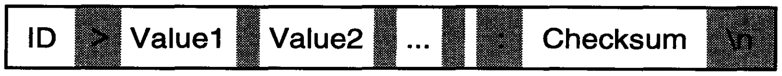


Figure 3-6: An Odyssey II network/modem communications packet. Sections variable in content and length are in white, static portions are in grey. The packet begins with an ID string followed by a ">" which identifies the type of information it contains. This is followed by data, with each value separated by a space. After the data, a ":" followed by a checksum string follows. The packet is terminated with a newline character.

essary for minor modifications to the mission. Instead, we decided on short control codes containing the important information describing the changes to the mission. Control commands were also checked. Again, because several observers could be controlling the vehicle, these commands were multicast over the network.

3.4 Software Implementation

3.4.1 Vehicle software

The vehicle software was updated with a routine called `comm` which ran as a separate process from the main vehicle code. The program `comm` could be started via the modems once the vehicle was in the water. It would then dynamically link to the vehicle data structure, giving it full access to the information contained within. Another file aboard the vehicle would specify which variables comm should send over the modem. These variables were typically heading, depth, altitude, and dead-reckoned position, although the software was written so the elements of the data structure to be sent were specified in a file.

Variables to be transmitted were incorporated into a packet as shown in Figure 3-6. An identifier string, followed by a ">", identified the contents of the packet. The names of the variables were prepended by a "`NAMES>`" string, while the variables themselves began with a "`VALUES>`" string. The variables were appended to this string, with a space between each variable. Finally, a ":" followed by a checksum generated from the string was appended to generate the final packet. A newline character would denote the end of the data packet. This entire packet was then sent through the modem at preprogrammed intervals, typically 1 Hz, although as can be seen in Figure 3-7, a lower refresh rate would have sufficed.

The commands coming over the modem had a similar arrangement. The program `comm` (running aboard the AUV) would receive the string and strip the checksum. It generated a checksum from the remaining string and compared it to the received checksum. If they did not match, an error message was sent over the modem. If the checksums matched, the received command was compared to a list of possible commands and appropriate action taken. For most of our experiments these commands were kept relatively simple, such as "abort" and "new setpoint." However, because `comm` had full access to the entire data structure, more complex modifications to the command stack could be made.

3.4.2 Shore station software

The modem receiving the signals from the vehicle modem was connected to the shore station computer — in our case a Silicon Graphics (SGI) Indy workstation. The shore station acted as a relay station, passing vehicle data from the modem to the network, and user commands from the network to the modem. A routine named `sgi_comm` checked the serial port for incoming data from the modem. Received data was tested for errors, and if error-free, was multicast over a free MBone IP address. Similarly, command signals coming over the MBone (same or different IP address) were checked for errors, and passed on to the modem if error free.

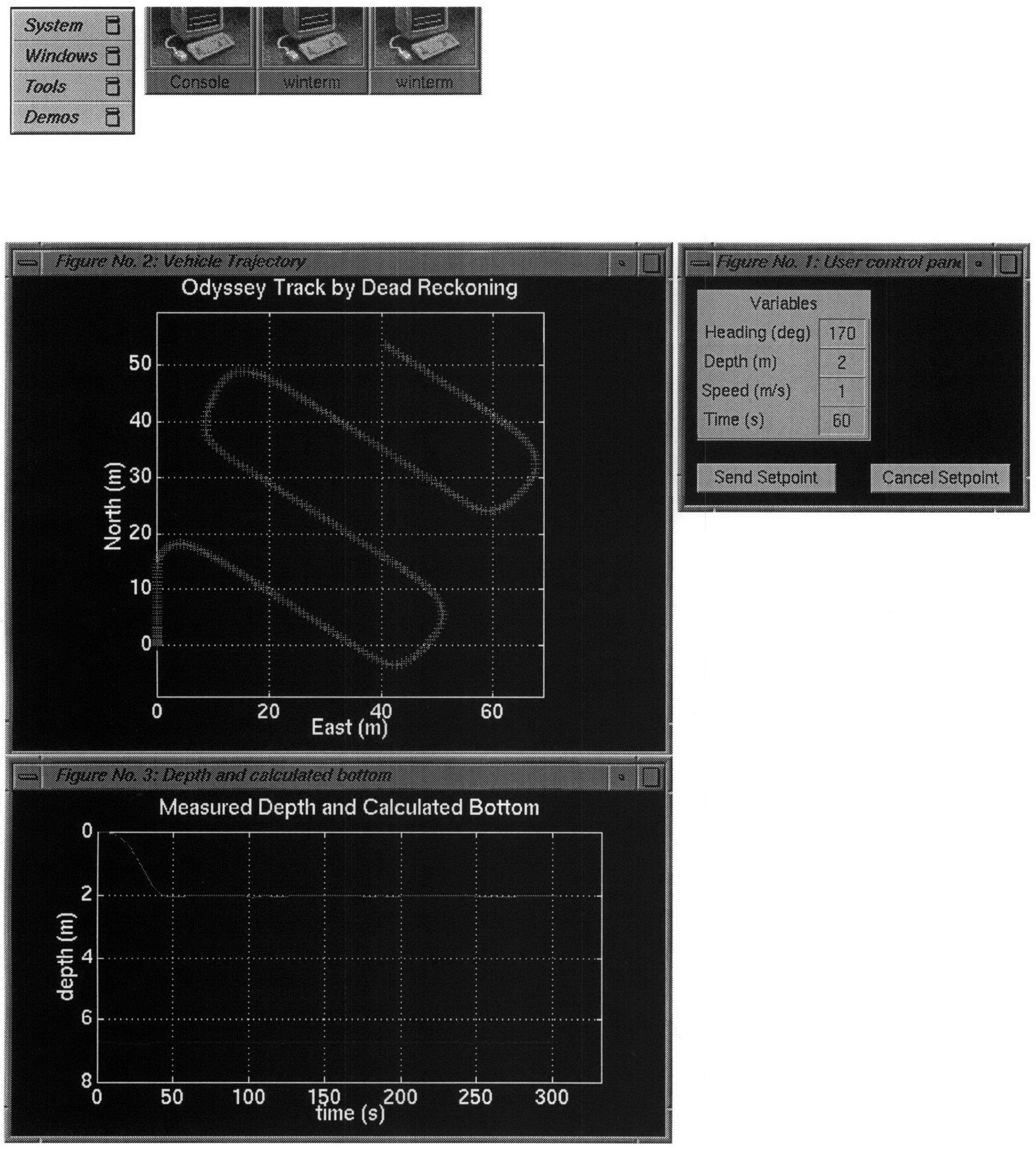
3.4.3 Control station software

The control station is where the user sits. For most of our tests this was also the shore station. However, from the software's point of view, the control station and shore station always look like different computers whether or not they are the same. The user ran Matlab on the control station, and a Matlab graphical user interface (GUI) script for monitoring and controlling the vehicle (Figure 3-7). This GUI script linked to an external C program which monitored the appropriate MBone IP address for data coming from the vehicle (via the shore station). The first packet sent from the vehicle contained initialization information, telling the Matlab script what variables would be sent and in what order. Subsequent packets were displayed on a north/east plot and a depth/altitude plot. These plots were dynamically updated as new data flowed in over the network. The script was simple and flexible enough that other variables could be shown in similar plots with a little modification to the code.

Another routine in the Matlab script accepted user input to abort the mission or reconfigure the current setpoint. These vehicle commands were encoded with a checksum and multicast over the network, where the `sgi_comm` program would receive and forward them to the vehicle. Because all the data transmission between the `comm` program and the user-end machine were multicast, it did not matter where the two machines were on the Internet, or even how many user-end machines there were. All user-end machines received and displayed data, and all could issue commands to the vehicle. For research purposes, the logistical and security problems this caused were not deemed important, but this area will need work in the future. In particular, care must be take to avoid conflicting commands sent by different users.

3.5 Summary

The Odyssey II software design was made flexible to allow easy incorporation of new sensors and instrument packages. This flexibility lent itself well to the addition of communications routines for monitoring and controlling AUV operations. A communications routine running aboard the vehicle was able to patch directly into the vehicle's data structure, giving it access to all the variables and commands used by the vehicle. In addition, the vehicle's layered control software design allowed communication of short, high level command codes to alter the vehicle's mission. Transmissions to and from the vehicle were relayed through a shore station computer to the Internet. In order to allow multiple users to observe and control the vehicle, these transmissions were multicast over the MBone. On the user's computer, a set of Matlab routines dynamically displayed data received from the vehicle, while allowing simple commands to be sent. Results of our experiments using this software are discussed in the next chapter.


System
Windows
Tools
Demos
Console
winterm
winterm
Figure No. 2: Vehicle Trajectory
Odyssey Track by Dead Reckoning
North (m)
East (m)
0
10
20
30
40
50
0
20
40
60
Figure No. 1: User control pane
Variables
Send Setpoint
Cancel Setpoint
Figure No. 3: Depth and calculated bottom
Measured Depth and Calculated Bottom
depth (m)
time (s)
0
2
4
6
8
0
50
100
150
200
250
300

Heading (deg)	170
Depth (m)	2
Speed (m/s)	1
Time (s)	60

Figure 3-7: The Matlab graphical user interface designed for monitoring and controlling the Odyssey II. While the plots used in the course of this research was limited to overhead dead reckoned and depth plots, any received data could be plotted using simple Matlab commands.

Chapter 4

Experiments

4.1 Introduction

This chapter describes our experiments controlling and monitoring an AUV with radio modems. It begins with a description of the additions and modifications to the vehicle hardware, as well as computers and network configurations. Next, the two major experiments are described, along with their goals, setup procedures specific to each experiment, and results. First our experiments on the Charles River, then our tests in an ocean environment at Buzzard's Bay are detailed. Vehicle position plots of several experiments are included to clarify the intent or finding of these experiments.

4.2 Hardware Implementation

Central to our communications experiments was an MIT Sea Grant Odyssey II AUV (described in Chapter 2). In addition to the software modifications described in Chapter 3, several new pieces of hardware were added or modified to the AUV. These included the modems, a box to protect the modem and battery from water, the tow float upon which the box was mounted, and a cable to connect the modem to the AUV. In addition to the AUV, we used one or two Silicon Graphics Indy workstations to act as the shore station and control user's workstation. For the Internet portion of these experiments, the computers were connected to MIT's local area network (and hence the Internet).

4.2.1 RF Modems

We used a pair of Proxim radio frequency (RF) modems capable of 121 kbps emulating either a serial modem (point-to-point) or a LAN (broadcast point-to-multipoint) [42]. For our operations we kept the modems in serial mode with a 19.2 kbps transmission rate. The omnidirectional antenna for the modem measured about 4 inches in length and mounted on one end of the modem. We added a penetrator to allow us to mount the antenna outside the watertight box. For the Charles River experiments the antenna was mounted outside with satisfactory results. During the Buzzard's Bay experiments, after complete failure to receive any signals from the modems, we moved the antenna inside the box with excellent results. To assess the effects of dropouts in modem transmissions, we turned off buffering in the modems during modem setup. However, the modems continued to buffer transmissions requiring us to send deliberately corrupt data to test our software error-checking.

4.2.2 Modem cable

The AUV's RF modem was mounted on a float towed behind the vehicle. A serial cable approximately 8 m long and 3 mm in diameter ran between the float and the vehicle. One end of this cable was wired to fit an extra Impulse wet-connector which in turn plugged into a sphere penetrator leading to a serial port on board the vehicle computer. The other end was a simple DB9 serial plug which went to the RF modem.

To remove the towing load from the serial cable, a tether made of low-stretch cord was connected in parallel and acted as the main load-bearing member. One end was tied to the lift point on the vehicle, while the other was attached to the float carrying the modem. The cord and serial cable were joined with tape and cable ties, with some slack in the serial cable to assure the cord took most if not all of the load. To prevent this tether from fouling the propeller, a small hole was drilled into the top of the rudder, and the tether was cable tied to the rudder. Small pieces of floatation were attached along the remaining tether to assure it wouldn't sink into the propeller while slack. For the experiments at Buzzard's Bay, the serial cable was replaced by a length of $\frac{3}{8}$ inch Impulse 8-conductor wet cable. This resulted in serious vehicle performance degradation which will be discussed in the following sections, but did not affect the outcome of our experiments.

4.2.3 Tow float

The tether was tied to the front of a 3 ft boogie board used as a float for the RF modem. The usual tow float used during shallow-water Odyssey operations on the Charles River had insufficient buoyancy and stability to hold the modem box upright. For our operations on the Charles River, the boogie board proved adequate as a tow float. During the Buzzard's Bay operations, due to the higher seas and additional weight of the Impulse wet cable, we had to tape the original tow float to the bottom of the boogie board for additional buoyancy and a stabilizing fin.

4.2.4 Modem box

Strapped and taped to the float was a watertight box measuring approximately 12 in by 9 in by 5 in high. While not pressure rated, it was sealed with a rubber O-ring at the seams. An additional layer of environmental tape (the same tape used to seal the vehicle spheres) helped to assure the watertightness of this seal. The top of the box screwed down into the O-ring with six long screws. In addition to the straps and tapes, this made opening and closing the box a laborious process taking several minutes. The box contained two penetrators — one for the serial cable and one for the antenna. The serial cable penetrator was originally a hollow plastic tube running through a hole in the box. A large amount of silicone, monkey gum, silicone grease, and tape was needed to make this penetrator watertight (splash-proof). For the Buzzard's Bay operations, we were able to replace this with a spare sphere penetrator with rubber O-ring. The antenna penetrator was a permanently mounted L-joint and sealed around the outside with silicone.

Mounted inside the box with velcro and cable ties were the RF modem and a sealed lead-acid battery ("Gel-Cell") for power. The power cable was wired with a switch mounted near the top of the box so that modem could be turned on or off without having to completely open the box. This allowed us to quickly reset the modem when something went wrong. It also let us reduce drain on the battery when the modem was not in use.

4.2.5 Shore station modem

An identical Proxim RF modem acted as the shore station modem, using either omnidirectional or directional (Yagi) antennas. As expected, the directional antenna provided

Table 4.1: Charles River Experiments, 31 October 1995.

Run	Mission	Antenna	Depth	Duration	Description
08	straight out	omni	2m	50s	Telemetry reception test
12	straight out	omni	2m	90s	Telemetry reception test
14	box	omni	2m	330s	Range test for omnidirectional antenna
15	box	omni	2m	450s	Range test for omnidirectional antenna
17	survey	yagi	2m	1700s	Range test for directional antenna
20	survey	yagi	2m	1750s	Range test for directional antenna

more range at the cost of directionality. However, for most of our operations we found the omnidirectional antenna to be sufficient.

4.2.6 Workstation

The shore station modem plugged into a Silicon Graphics Indy (SGI) R4000 running IRIX 5.3. In theory, any computer with a serial port capable of generating and receive messages compatible with the command set and vehicle telemetry would have sufficed. However, since we planned to control and monitor the vehicle over a network and the Internet, we picked a Unix workstation as the best development platform for generating portable code utilizing both the network and serial ports.

4.3 Charles River Experiments

For the Charles River experiments in October and November of 1995, we placed the Unix workstation and the second modem with directional antenna at the dock of the MIT sailing pavilion. An ethernet drop at the dock provided the connection to the Internet. The user interface package usually ran on this workstation, but data was always multicast over the MBone (deliberately limited to only MIT's network for these experiments), so anyone on MIT's network could have observed the incoming data at any time.

4.3.1 31 October 1995

An initial series of trials on October 31 (Table 4.1) only received telemetry from the vehicle without attempts at control. Omitted run numbers indicate unpowered tests of the vehicle near the surface, simulated missions, and missions which were aborted soon after launch for a variety of reasons. Because a navigational array was not deployed in the water

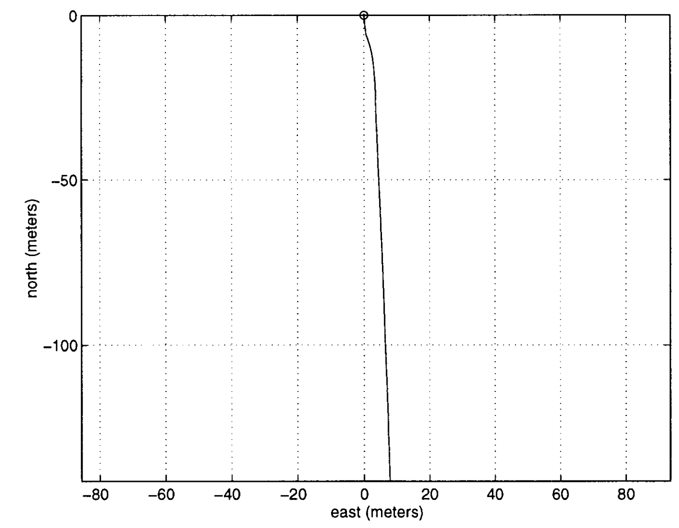


Figure 4-1: Dead-reckoned vehicle trajectory for Charles River experiment 31 October 1995, mission 12.

at the time of these experiments, all positions are dead-reckoned assuming a constant vehicle velocity of 1.35 m/s.

We had extensively tested our software in the lab with a vehicle simulator so missions 8 and 12 (Figure 4-1) were simply to test that the software was working correctly with a real vehicle. After determining that we were receiving vehicle data telemetry, missions 14 and 15 tested the range of the modems using the omnidirectional antenna (Figure 4-2). The modem's range seemed to extend well past 200 m, but there were periods of 5 to 20 seconds during which we received no telemetry.

We repeated these experiments with the directional (Yagi) antenna using a survey pattern to determine both range and directionality of coverage, shown in missions 17 (Figure 4-3) and 20 (Figure 4-4). As expected, coverage extended further than the omnidirectional antenna, with dropoffs towards the sides. However, we still experienced periods of 5 to 20 seconds during which we received no telemetry.

Although we had explicitly turned off data buffering in the modems, it became apparent the modems were still buffering and resending any erroneous data. This had the effect of making the plots in Matlab useless for analyzing these dropouts because the missing data were eventually sent, making the Matlab plots appear error-free upon later analysis. Thus we had to manually record the times the dropouts occurred. Analysis failed to yield a pattern, but our primary suspect at the time was a faulty modem cable which may not have

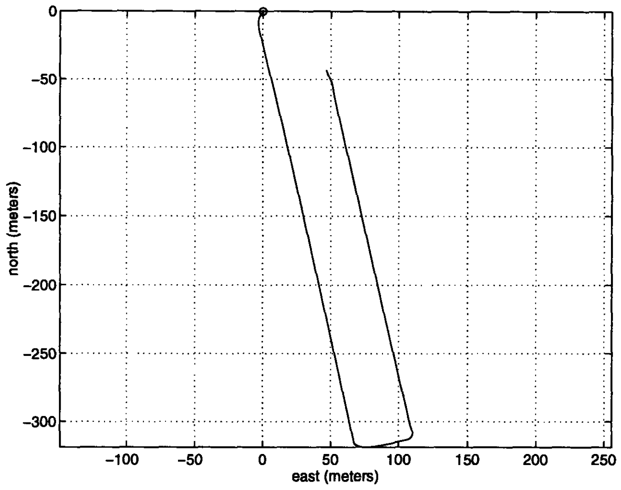


Figure 4-2: Dead-reckoned vehicle trajectory for Charles River experiment 31 October 1995, mission 15.

Table 4.2: Charles River Experiments, 7 November 1995.

Run	Mission	Antenna	Depth	Duration	Description
02	survey	yagi	2m	665s	Test of equipment
08	straight out	yagi	1m-3m	110s	Abort command test, depth change test
10	box	yagi	1m-3m	450s	Abort command test, depth change test
12	box	yagi	1m-3m	421s	Abort command test, depth change test
14	box	yagi	1m-3m	421s	Abort command test, depth change test
15	straight out	yagi	1m-3m	107s	Depth change test, propeller fouled
17	straight out	yagi	2m	107s	Mission aborted
18	straight out	yagi	1m-2m	205s	Commanded depth change, mission aborted
19	commanded	yagi	2m	391s	Commanded heading changes
20	survey	yagi	2m	979s	Range test

been entirely waterproof. This theory was rejected after the Buzzard's Bay experiments, but because the communications link was so unreliable, we decided to use the directional antenna for future experiments, in the hope that the additional gain would help prevent some of these dropouts.

4.3.2 7 November 1995

Our experiments on 7 November 1995 represented our first attempts to send commands to the AUV during a mission. Mission 2 was a short survey to determine that all equipment and software was functioning properly, and to test if the dropouts we had had previously encountered were still present. They were, and without any real pattern, we decided to

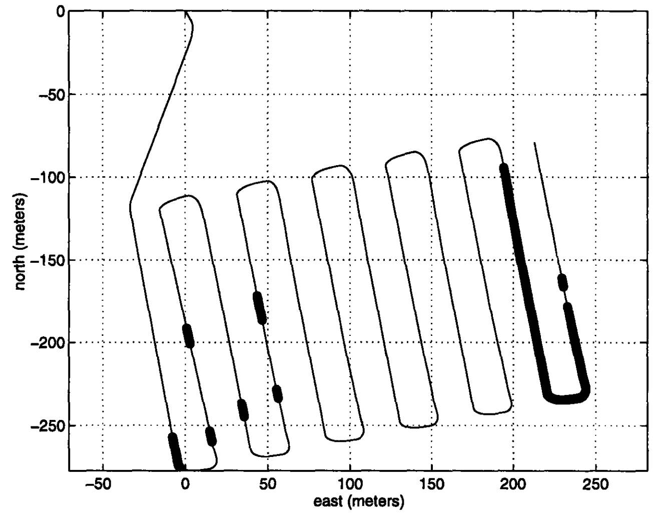


Figure 4-3: Dead-reckoned vehicle trajectory for Charles River experiment 31 October 1995, mission 17. Broad portions indicate times in which we observed RF modem dropouts.

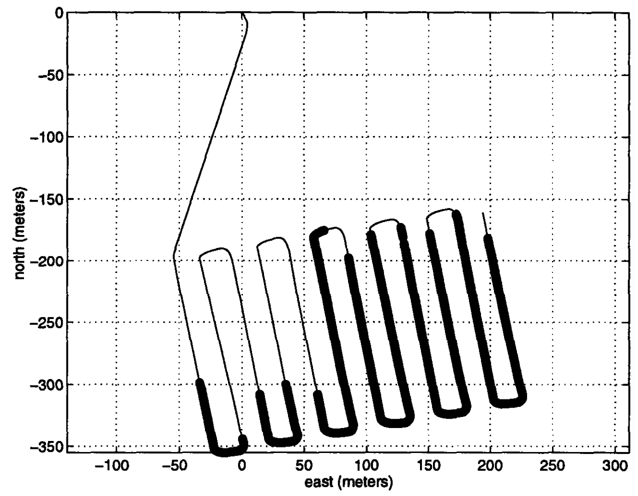


Figure 4-4: Dead-reckoned vehicle trajectory for Charles River experiment 31 October 1995, mission 20. Broad portions indicate times in which we observed RF modem dropouts. Note the reduced range compared to mission 17, especially near the end. This was probably due to the batteries being nearly drained.

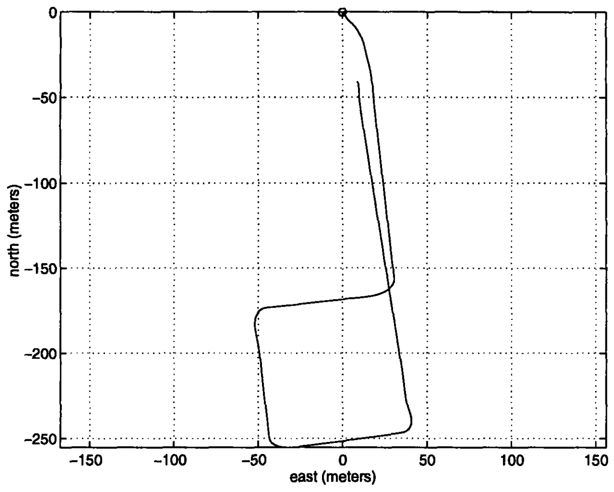


Figure 4-5: Dead-reckoned vehicle trajectory for Charles River experiment 7 November 1995, mission 12.

ignore them for the time being. Missions 8, 10, 12 (Figure 4-5), and 14 represented tests of our abort command, which immediately ends the mission aboard the vehicle computer.

Satisfied that the abort command was working, we tried changing the depth of the vehicle during missions 15 and 18. The modem cable ended up being fouled by the propeller, so we redistributed the floatation foam to allow greater clearance. Mission 19 (Figure 4-6) represents the first time the vehicle was steered during a mission by sending a new commanded heading. We were able to steer the vehicle in a rough rectangle, send it back towards the dock (reducing the time needed to retrieve the vehicle), then abort the mission.

With daylight running out and the modem battery voltage running low, we decided to investigate the possibility that the dropouts were somehow related to battery voltage. We sent the vehicle on another survey mission (Figure 4-7) far out into the river. Due to the drop in battery voltage, we lost contact with the modems at a shorter range than on the previous missions, and being unable to send an abort command, were forced to manually abort the mission by taking a boat out and grabbing the tow float (hence the erratic dead-reckoned path in the figure). Since a drop in battery voltage seemed to cause an overall reduction in modem range instead of an increased number of dropouts at close range, we dismissed it as the cause of the dropout problem.

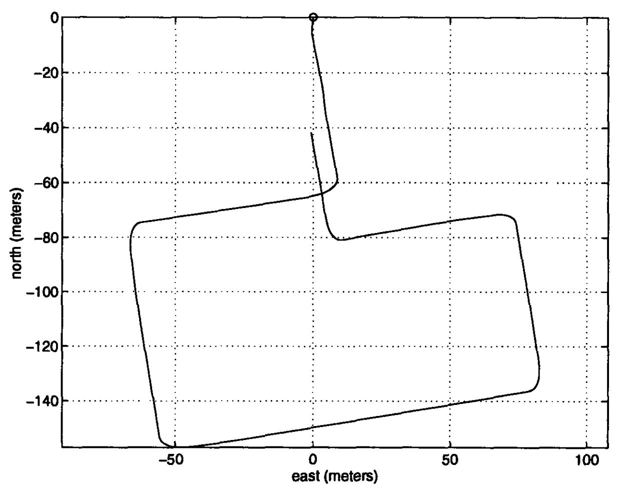


Figure 4-6: Dead-reckoned vehicle trajectory for Charles River experiment 7 November 1995, mission 19.

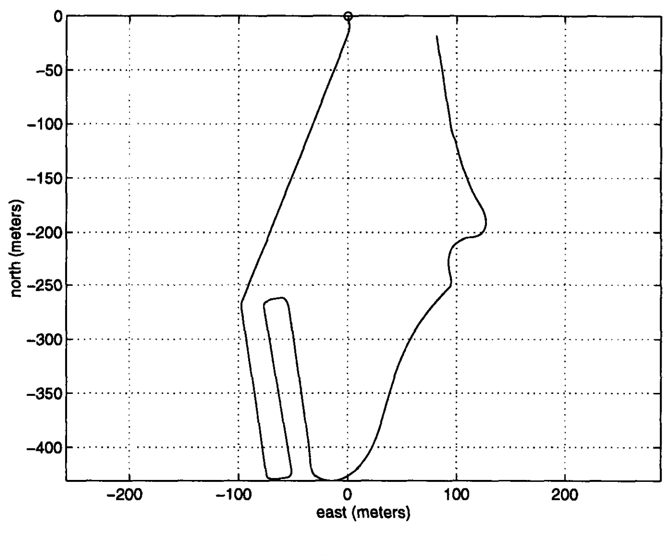


Figure 4-7: Dead-reckoned vehicle trajectory for Charles River experiment 7 November 1995, mission 20. The erratic path at the end was due to aborting the mission by grabbing the tow float.

Table 4.3: Charles River Experiments, 16 November 1995.

Run	Mission	Antenna	Depth	Duration	Description
10	commanded	yagi	2m	30s	Propeller fouled, mission aborted
14	commanded	yagi	1m–3m	330s	Commanded heading changes to form box
16	commanded	yagi	1m–3m	330s	Commanded heading changes to form box
20	commanded	yagi	2m	361s	Commanded heading changes to form box

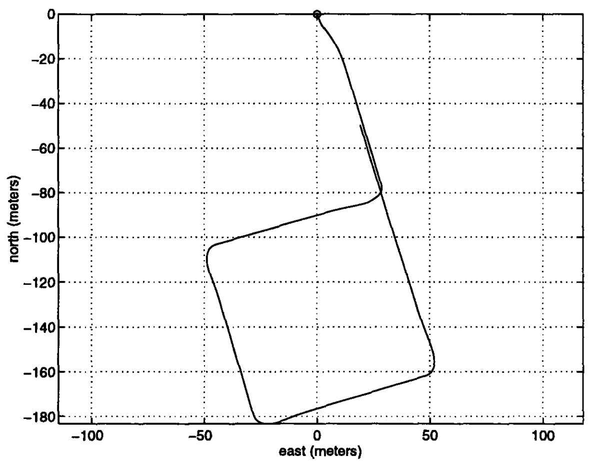


Figure 4-8: Dead-reckoned vehicle trajectory for Charles River experiment 16 November 1995, mission 14.

4.3.3 16 November 1995

The experiments on 16 November 1995 were further tests sending the vehicle new commanded headings over the modems. Also implemented for these missions was rudimentary error-checking of the commands and data sent over the modem using a checksum. However, since the modems seemed be error-checking and buffering data on their own, this improvement had no noticeable effects other than peace of mind. Missions 14 and 20 (Figures 4-8 and 4-9) demonstrate some of the simple boxes made by steering the vehicle.

4.3.4 21 November 1995

On 21 November 1995, due to a near-disaster, we implemented another safety feature in the vehicle control software. We were also able to control and monitor the vehicle from the MIT Sea Grant AUV Lab over the Internet for the first time.

Figure 4-10 shows mission 3 in which we were commanding the vehicle in patterns other than simple boxes. Towards the end of the mission, we noticed a crew boat approaching.

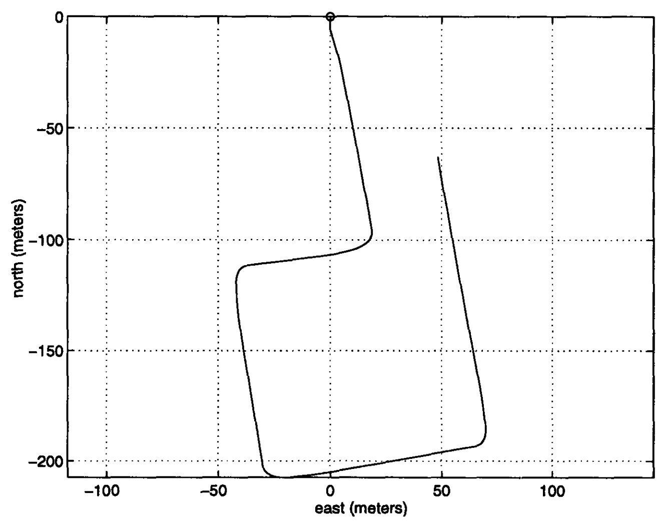


Figure 4-9: Dead-reckoned vehicle trajectory for Charles River experiment 16 November 1995, mission 20.

Table 4.4: Charles River Experiments, 21 November 1995.

Run	Mission	Antenna	Depth	Duration	Description
03	commanded	yagi	2m	819s	Nearly steered into crew boats
08	circling test	yagi	2m	300s	Test of circling routine
12	commanded	yagi	2m	600s	Commanded heading changes with circling
15	Internet control	yagi	2m	817s	Commanded heading changes via Internet
17	Internet control	yagi	1.5m–2m	895s	Commanded heading changes via Internet
18	Internet control	yagi	1.5m–2m	835s	Commanded heading changes via Internet

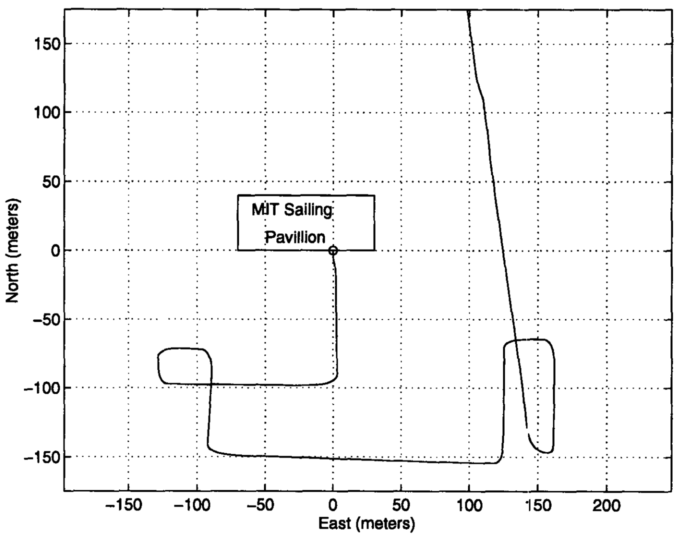


Figure 4-10: Dead-reckoned vehicle trajectory for Charles River experiment 21 November 1995, mission 3.

Judging the crew boat to pass between us and the vehicle, we commanded the AUV to head away from us. Shortly afterwards, we realized we had erred and the vehicle was now headed into the path of the crew boat. We sent another command to the vehicle to head back towards us. Unfortunately, a communications dropout occurred at this time and the vehicle continued towards the crew boat. It passed well in front of the crew boat, only to turn back around as the modems had buffered the turn command during the dropout and had finally managed to send the command through. At this point we got in our boat and grabbed the float by hand, stopping the vehicle. Since the vehicle was still powered, the dead-reckoned position, which assumes a constant 1.35 m/sec velocity, shows the vehicle running straight into the wall at full speed. If we had not grabbed the vehicle by hand, and if a dropout had blocked our abort command, the vehicle would have run into the wall as its dead-reckoned trajectory indicates.

This experience taught us that any command sent to an autonomous vehicle should always be assumed to be the last command it will ever receive, and a routine command right now may be a disastrous command a short time later. In response, we implemented a circling routine. A fixed amount of time (60 seconds in these runs) after receiving a new command, the vehicle loiters in a tight circle to await a new command. In the future, the vehicle could default to some other preprogrammed behavior. In the presence of navigation equipment capable of locating the vehicle's position, one approach is to use a preprogrammed boundary

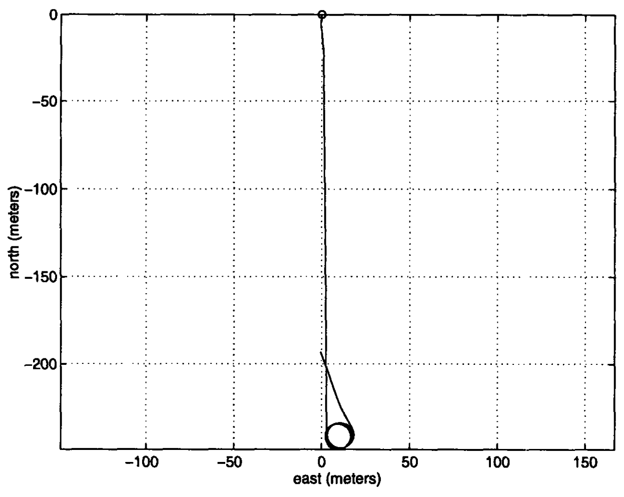


Figure 4-11: Dead-reckoned vehicle trajectory for Charles River experiment 21 November 1995, mission 8.

which the vehicle is not to cross. Figure 4-11 shows our first test of the circling routine.

With the circling routine in place, we attempted to operate the vehicle from a remote location. One person monitored the vehicle from the dock, while another commanded the vehicle over the Internet from the Sea Grant laboratory several blocks from the river. Our only major problem was that MIT Network Services had shut down the MBone repeater for the sailing pavilion because it had not been used for a month since we had first requested it be activated. Since our multicast code sends all data and commands over the MBone, without the repeater, no multicast communications between the dock and lab were possible. Because we did not know the repeater had been shut down, we spent several hours stepping through our code trying to locate the problem.

Once the multicasting repeater was reactivated, both the dock computer and laboratory computer were able to simultaneously monitor and control the vehicle during a mission (Figures 4-12, 4-13, and 4-14). All the commands were sent from the laboratory computer except for the abort command to end each mission. However, because the laboratory was out of sight of the actual vehicle and the river, we had to rely extensively on telephone communications with the person at the dock to verify the vehicle was doing what our display said it was doing. Because of this lack of feedback, we limited most of our commands to the East-West direction, minimizing the risk of sending the vehicle into the dock or a wall. This problem should disappear with time as users gain confidence in the software and the correspondence between the vehicle's actual position and that shown on the GUI.

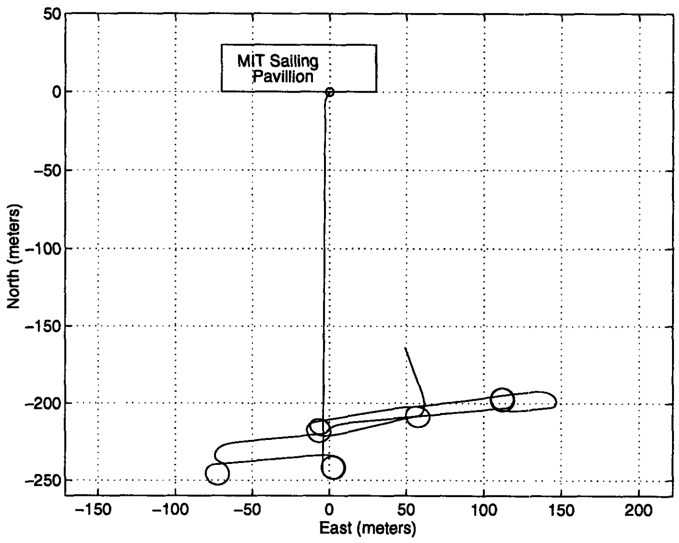


Figure 4-12: Dead-reckoned vehicle trajectory for Charles River experiment 21 November 1995, mission 15.

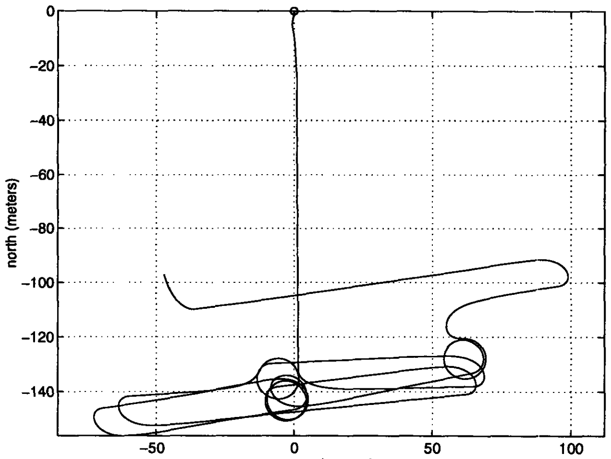


Figure 4-13: Dead-reckoned vehicle trajectory for Charles River experiment 21 November 1995, mission 17.

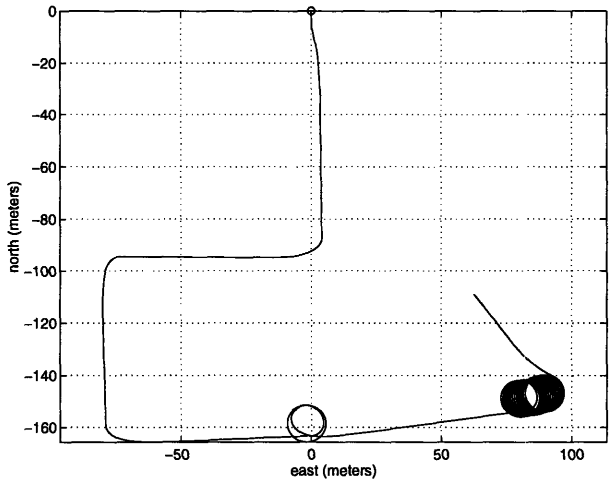


Figure 4-14: Dead-reckoned vehicle trajectory for Charles River experiment 21 November 1995, mission 18.

Table 4.5: Buzzard's Bay Experiments, 9 April 1996.

Run	Mission	Antenna	Depth	Duration	Description
25	commanded	omni	2m	1200s	Test with antenna inside box
26	commanded	yagi	2m	1001s	Tried to spell MIT

4.4 Buzzard's Bay Experiments

During the Buzzard's Bay experiment in March-April 1996, we tested telemetry and control of the vehicle in salt water at an actual experiment site. Operations were based from the *R/V Diane G*, with an SGI workstation acting as a node and shore station as with the Charles River experiments. Because the acoustic modems were not integrated yet, we continued to use the radio modems. After initial trials resulted in no telemetry whatsoever, we ordered a waterproof cable to replace our handmade cable. When this failed to correct the problem, we tried placing the float mounted antenna inside the watertight box, as it was the only remaining hardware still exposed to water. We received strong telemetry afterwards, and even the dropout problem we experienced on the Charles River disappeared.

Unfortunately, the replacement cable was much thicker and heavier than our handmade cable, and combined with the heavier seas on the bay, caused the tow float to become unstable, flipping it over several times. The normal Odyssey II tow float has a stabilizing fin, and we were able to tape this float onto the radio modem float. While this combined with the additional modem cable thickness drastically increased the drag on the vehicle,

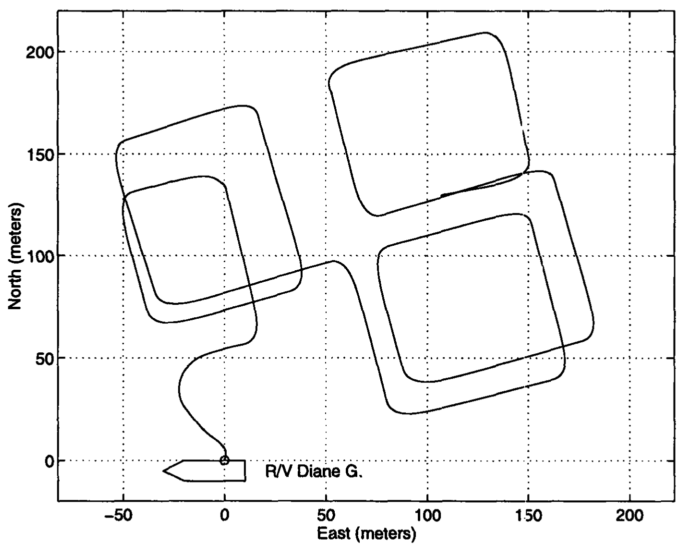


Figure 4-15: Dead-reckoned vehicle trajectory for Charles River experiment 9 April 1996, mission 25.

the vehicle was still able to maintain depth and operate normally albeit slowly.

Improvements to the software included full cyclic redundancy check (CRC) error checking, and the ability to request and modify any variable in the vehicle computer's data stack. Although the *R/V Diane G* had no connection to the Internet, the software was still configured to multicast, and had there been an Internet connection or another computer listening on the ship's local area network, it would have been able to simultaneously observe the experiment with us.

Figure 4-15 shows our first mission with the antenna mounted inside the watertight box. After verifying that we were getting good telemetry, we decided to test the range of the omnidirectional antennas now that we had a dropout-free link. The further boxes represent our steering the vehicle further and further away from the ship. No dropouts or loss of signal due to range was detected, and we decided to do something more productive.

Figure 4-16 shows the vehicle steered from the *R/V Diane G* to spell most of the initials of MIT. Due to the range this feat required, we switched to the directional antenna, which also functioned without any dropouts. Unfortunately, an operator error and rapidly approaching Nor'easter forced us to abort before completion.

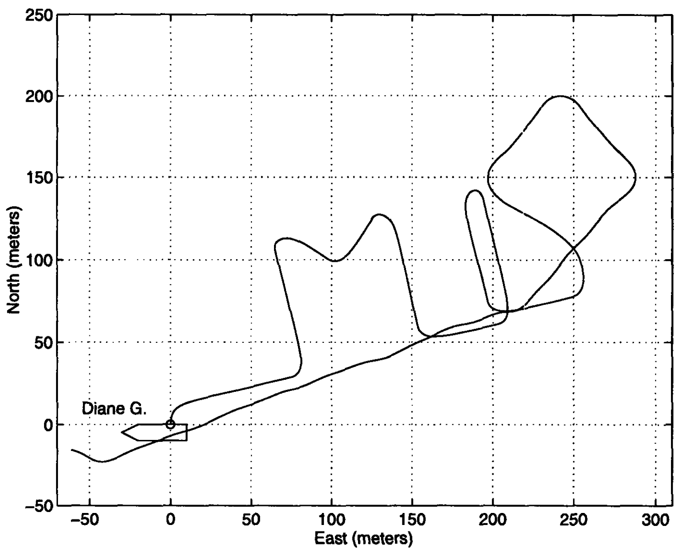


Figure 4-16: Dead-reckoned vehicle trajectory for Charles River experiment 9 April 1996, mission 26.

4.5 Summary

This chapter has described our remote control and monitoring experiments of an AUV on the Charles River and in Buzzard's Bay. From these experiments, we have determined several issues to deal with in future experiments. Communications with the vehicle are unreliable, and time delays can cause commands to have unintended effects. The vehicle must have enough onboard intelligence to extract itself from dangerous situations caused by partial or erroneous commands. While the dropouts we encountered were random, shadow zones producing similar effects can be expected with acoustic communications. Because multicasting is not yet widely used throughout the Internet, certain problems can arise which are difficult to diagnose unless one is well versed in the low-level network operations of multicasting. Confidence in a new system with no direct visual feedback is low and users will need time to acclimate. Problems associated with communications in a multiple vehicle environment are discussed in the next chapter. And further exploration of all these problems and possible solutions are covered in Chapter 6.

Chapter 5

Multiple AUV Communications Management

5.1 Introduction

The previous chapters have explored the issue of communicating with a single AUV during a mission. As mentioned in Chapter 1, one type of application involves the deployment of multiple AUVs in an experiment site. Communicating with multiple moving vehicles is a complex problem due to the signal propagation delays and possibility of signals interfering with each other. Some of these issues are identical to those encountered in radio and electronic network based multi-node communications, and various solutions to those problems are presented and discussed. However, the properties of the ocean as an acoustic communications channel are rather unique, and the radio and network solutions are not entirely applicable. Strategies to address problems specific to the ocean are discussed along with their potential benefits and drawbacks.

5.2 The Multiple AUV Communications Problem

In communications with a single AUV in the operational area, several implicit assumptions were made. The AUV and shore station (or node) were assumed to be the only high-power acoustic sources in the immediate location and in the frequency bandwidth assigned for acoustic communications. Because the bandwidth required to transmit the AUV's position was so small, the shore station had the luxury of receiving regular position updates from the

AUV as it moved through the water. The AUV always had the most complete information about its surroundings because it was the only source of new information. With a single vehicle, position fixes relative to a global coordinate system can be made using methods more accurate than traditional triangulation. [6]. However, many important missions for AUVs will require the use of multiple vehicles. Hence, we decided to continue our work by addressing the issue of communications between multiple vehicles.

If multiple vehicles are to operate together during a mission, the above assumptions cannot be made. Multiple acoustic sources are in the water, and two or more transmissions may arrive at a receiver simultaneously (collide), thus garbling their contents (Figure 5-1). Preventing these collisions involves some reduction in the bandwidth available for any one AUV to transmit information. This has several implications for monitoring AUVs (by the user or by the vehicles themselves). The navigation problem increases in complexity [2] due to potential collisions of navigation signals, and all vehicles may not know their current position relative to a global coordinate system at a given time. For the vehicles to operate together on a common goal, they must share the information they gather. Because of the bandwidth limitations, this information may not be distributed in a timely manner. In particular, the position may be inaccurate or unimportant enough to be impractical to transmit. This may further reduce the available bandwidth, thus compounding the problem even more.

5.2.1 Criteria

In light of these difficulties, several criteria can be used to appraise the effectiveness of a multi-vehicle communications scheme:

- Utilization of the communications channel
- Maximum bandwidth available to one vehicle
- Collision rate
- Message queueing delay
- Scaling with number of vehicles

Utilization of the communications channel is the most obvious measure. As more and more information needs to be exchanged, transmissions should be occurring more frequently.

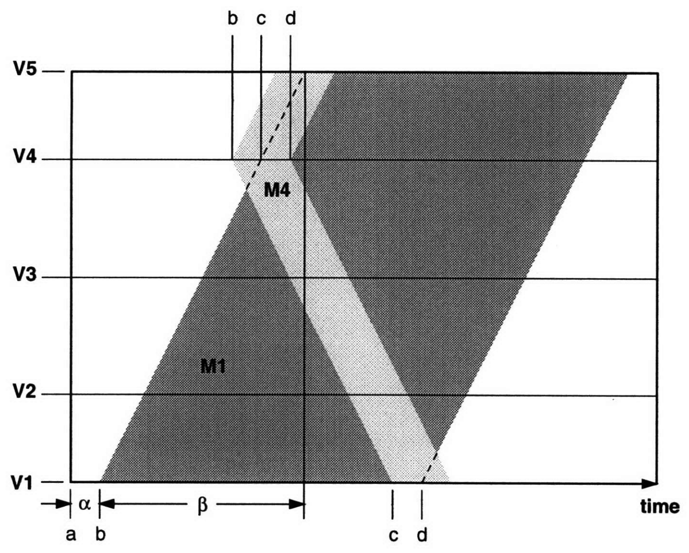


Figure 5-1: Transmission collision. a) Vehicle sends message to its modem. b) Modem begins transmitting. c) Collision with message from another modem occurs. d) Modem stops transmitting. In time sequence, vehicle 1 sends a message to its modem for transmission. After a processing delay, α, the modem begins transmitting message M1, which will take time β to propagate to the furthest vehicle. While vehicle 1 is transmitting, vehicle 4's modem begins sending a message M4. Shortly after, vehicle 4 detects a collision and stops transmitting. However, vehicle 1 does not yet know there is a collision and continues to transmit. Finally, M4 reaches vehicle 1, which realizes there is a collision and stops transmitting.

In a perfect scheme, the channel would be saturated with transmissions 100% of the time. The portion of the channel utilized can be viewed overall, or on a per vehicle basis. Another criteria is the maximum bandwidth available per vehicle. Due to buffering or collision-preventing requirements, some communications schemes must devote portions of the available bandwidth to non-data transmissions. Yet another criteria is the number of collisions relative to channel utilization. Collisions garble transmissions, forcing the information to be rebroadcast, thus wasting bandwidth. 100% utilization of a channel is useless if none of the information being transmitted is correctly received by the other vehicles. Because the channel must be shared, some vehicles will have to wait to transmit their information. While waiting or transmitting, the vehicle may collect more data which must be queued. The expected duration of these delays can help us assess whether the communications scheme is quick enough to be useful. Finally, measurements for each of these criteria will change as the number of communicating vehicles increases. Thus how a particular communications scheme scales with the number of vehicles is an important criteria.

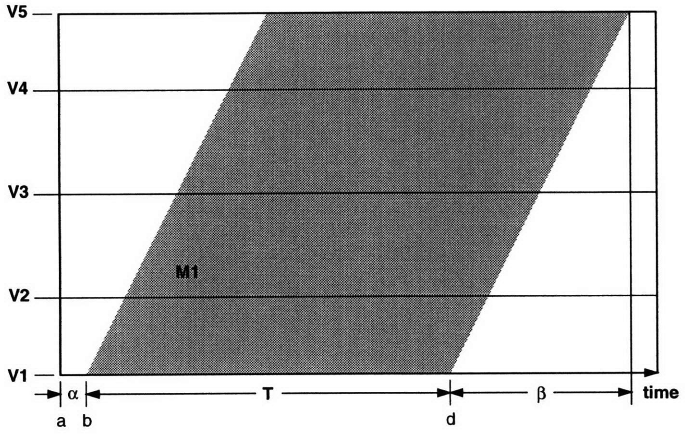


Figure 5-2: Delays in a transmission signal. a) Vehicle queues data. b) After a processing delay, α, the vehicle's modem begins to transmit the data, M1. c) After a time, T, called the transmission delay, the modem finishes transmitting. d) The data reaches the furthest vehicle or the edge of the deployment area after a propagation delay, β.

5.2.2 Definitions

In order to measure these criteria, several definitions and assumptions must be made to state the problem mathematically. Figure 5-2 shows a generic transmission from a vehicle into the communications channel. The parameter α represents a processing delay plus a queuing delay. The processing delay is the amount of time it takes for the onboard computer and modem to convert information into acoustic signals. The queuing delay is the time the vehicle must wait for the channel to become free. After that is T, the transmission delay — the time needed to transmit the information. Following completion of the transmission is β, the propagation delay, which is the time required for the signal to propagate from the vehicle's current position to all other vehicles (if their positions are known), or throughout the entire operating area. For acoustic transmissions, it is directly tied to the speed of sound. For each vehicle receiving a transmission, there is also a processing delay, γ, which is the time needed for the modem to extract the original information from the received acoustic signal.

For simplicity, n vehicles are assumed to be operating in two dimensions in the same horizontal plane, within a circular region of diameter R. For clarity, figures will assume the vehicles are operating in a line with each other. On occasion, subscripts will be used to denote values for a particular vehicle. Sound speed, c, is assumed to be constant, and the effects of multipath are not considered except implicitly as a contribution to γ, the received signal processing delay. Furthermore, because α and γ are hardware-dependent, none of our calculations explore their ramifications in detail. For brevity, we will use α to denote all delays other than transmission and propagation. We will also assume that all signals are received (although they may be garbled) and all collisions are detected.

The total bandwidth of a channel will be denoted by B. For slotted communications, in which time is divided into regular slots defining when a transmission can begin, S is the duration of a time slot. Data packets needing to be transmitted are assumed to arrive according to independent Poisson processes where λ is the overall arrival rate, and λ/n is the arrival rate at one vehicle.

Our evaluation criteria are U, the fraction of the transmission channel utilized for transmitting data; U_i, the fraction of the transmission channel utilized by a single vehicle; B_i, the bandwidth available to a single vehicle; C the fraction of transmitted packets that collide with other packets; and W, the queueing delay before a packet is transmitted.

5.3 Related Problems

The problem of multiple nodes communicating over a shared channel is not unique to ocean acoustics. These systems are referred to as multiaccess channels, and a variety of schemes have been developed to classify and address their differences. Examples of other multiaccess channels include satellite channels and multitapped buses such as computer networks. Satellites often receive data simultaneously from multiple sources over a single antenna. The bandwidth of the antenna must be allocated among the transmitters for the satellite to receive all the incoming information. Computer networks allow multiple computers to communicate with each other. However, should several computers attempt to communicate at once, their signals will interfere, causing a collision resulting in a failure of any computer to receive the signal.

Solutions to the multiaccess problem fall roughly into two categories [10]. One is the scheduled approach, in which transmissions are ordered in such a way that they do not interfere with each other. Multiplexing, whether in the time or frequency domains, falls under this category. Each node (vehicle) transmits in its assigned time or frequency. Since the other nodes are aware of when or how the other nodes will transmit, there is no interference between nodes.

The other approach is a "free-for-all" method in which nodes transmit their information as soon as they can. This invariably leads to collisions, requiring that the information be retransmitted. Statistical modeling methods can be used to analyze how often these collisions occur and how they affect the system. This allows a particular system to be tuned to provide optimal performance.

A variety of additional strategies have been developed to address shortcomings of these two methods. For example, carrier sensing is often used to reduce collisions in a "free-for-all" system by suppressing nodes from transmitting while one node is using the channel. Unfortunately, carrier sensing is primarily useful when the propagation delay is very small, and hence is inapplicable to our problem. On the other hand, reservations are useful for systems with large propagation delays like ocean acoustic channels. In a reservation system, each node preallocates as much channel bandwidth as it needs to transmit its information. Since the reservations are smaller in size than the actual data, they can be sent more quickly and with less risk of collisions.

5.4 Communications Schemes

5.4.1 Multiplexing

Multiplexing splits up the available bandwidth so that each vehicle can transmit for a set amount of time during which its signal is guaranteed not to collide with another vehicle's transmissions. For multiplexing with n vehicles and available bandwidth, B, in an optimum case each vehicle has B/n fraction of the bandwidth available for its transmissions. In reality, some bandwidth must be sacrificed as a buffer to clearly separate the sections allocated to each vehicle. The two most prevalent forms of multiplexing are time domain and frequency domain multiplexing, which are described below. A third scheme, code domain multiplexing, exists in which orthogonal code sequences are assigned to each node. We did not investigate code domain multiplexing at this time. Multiplexing communications schemes do not suffer from collisions.

Time domain multiplexing

Time domain multiplexing (TDM) divides the available bandwidth into time slots (Figure 5-3). Each vehicle is assigned a slot during which it can transmit. A short time buffer should

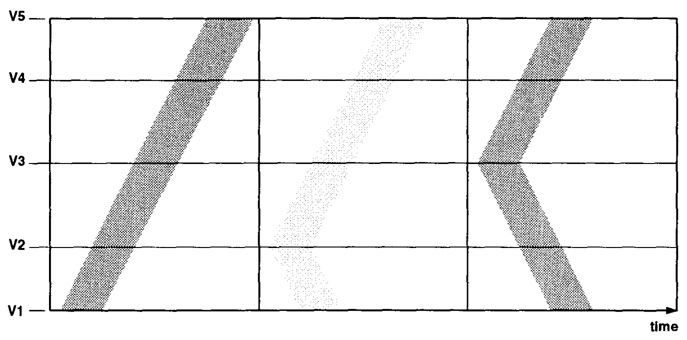


Figure 5-3: Time domain multiplexing. Time is divided into regular slots of duration $\alpha + T + \beta$, the sum of the processing, transmission, and propagation delays. Each vehicle is assigned a different slot in which it is allowed to transmit messages. In the first slot, vehicle 1 transmits. Vehicle 2 transmits next, vehicle 3 transmits third, and so on. When all vehicles have transmitted, the time slots repeat.

be added to insure separation of slots, but we will assume the buffer time is shorter than the processing delays in α. When all vehicles have transmitted, the cycle repeats. In this simplest implementation the length of a time slot, S, is

$$S = \alpha + T + \beta \tag{5.1}$$

of which only T represents utilization of the communications channel. In order to base our time slots on a clock synchronized between all vehicles, we have assumed β to be based on the largest possible distance between any two vehicles. That is,

$$\beta = \frac{R}{c} \tag{5.2}$$

Hence,

$$U = \frac{T}{S} = \frac{T}{\alpha + T + \frac{R}{c}} \tag{5.3}$$

Because time slices are divided evenly between vehicles, the utilization per vehicle, U_i, is just $\frac{U}{n}$

$$U_i = \frac{T}{n(\alpha + T + \frac{R}{c})} \tag{5.4}$$

While a vehicle is transmitting, the maximum bandwidth available to it is the entire bandwidth of the channel. $B_i = B$.

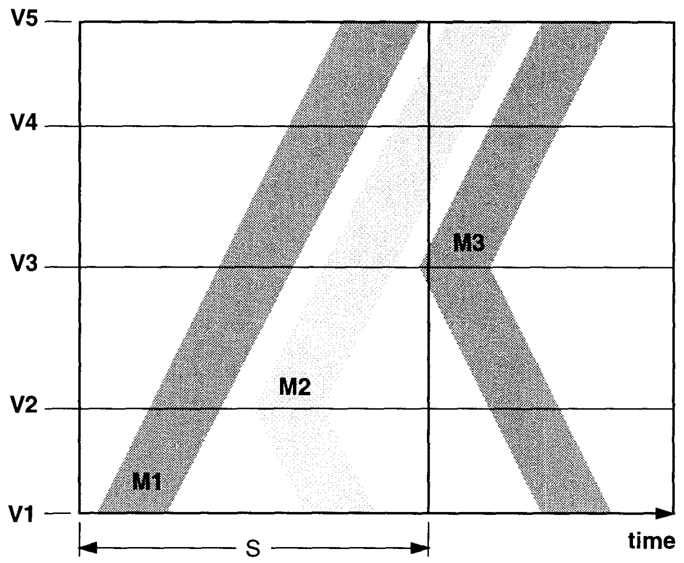


Figure 5-4: Optimized time domain multiplexing. Vehicles transmit in succession as with normal TDM, but instead of waiting for the full propagation delay, β, each vehicle begins transmitting when it detects the previous vehicle has finished. This eliminates wasted idle time while still avoiding collisions. The third vehicle has already begun its transmission by the time a normal TDM time slot, S, has passed.

Because vehicles must wait their turn before transmitting, a substantial queueing delay can develop. For TDM, the queueing delay is

$$W = \frac{n}{2(1-\lambda)} \tag{5.5}$$

time slots [10]. The delay increases linearly with n while the bandwidth utilized by a vehicle has an inverse relationship with n.

For large propagation delays, most of the time is spent waiting for the channel to become clear from propagation. Because β is large for most underwater acoustic transmissions, a more complex scheme could reduce this delay by having the next vehicle transmit as soon as it detects that the previous vehicle has finished (Figure 5-4). This method assures there are no collisions in much the same way paper cones will stack even their points do not line up. The location of the vehicles then comes into play, as the propagation delay then becomes the time for the signal to travel from vehicle V_i to V_{i+1}. Then

$$\beta = \frac{R_{i,i+1}}{c} \leq \frac{R}{c} \tag{5.6}$$

where $R_{i,i+1}$ is the distance between vehicles V_i and V_{i+1} The other criteria are identical to

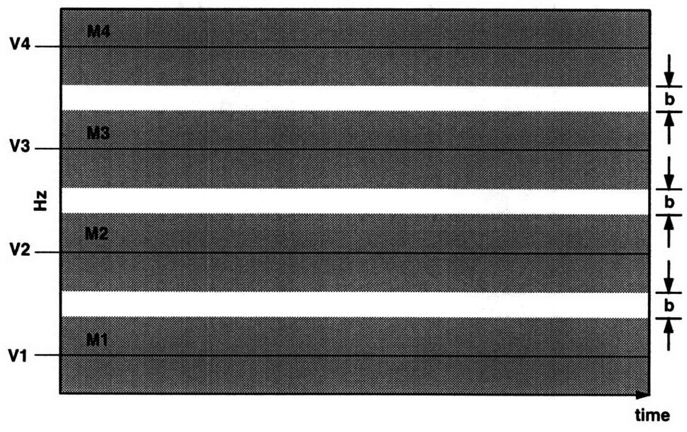


Figure 5-5: Frequency domain multiplexing. Each vehicle is assigned a different frequency band. The different bands are separated by a buffer, b to prevent interference due to doppler or other frequency shifts. Each vehicle owns its band entirely, and can transmit at any time without fear of collisions. However, the bandwidth available to a single vehicle is greatly reduced.

regularly spaced TDM. The duration of a time slot is then $S = \alpha + T + \frac{R_{i,i+1}}{c}$.

An obvious way to further refine this method would be to minimize the total time waiting for propagation between vehicles by minimizing $\sum_{i=1}^{n-1} R_{i,i+1}$, the sum of the distances between consecutive vehicles. Unfortunately, this is the classic traveling salesman problem of minimizing distance traveled, which rapidly becomes computationally intensive for large n even if we were to assume we knew each vehicle's position at all times. Another potential problem with this method is maintaining synchronization or reinitializing the system should a vehicle become disabled or enter a shadow region.

Frequency domain multiplexing

Frequency domain multiplexing (FDM) divides the available bandwidth into frequency slots, each assigned to a vehicle (Figure 5-5). It has an advantage over TDM in that vehicles do not have to wait their turn before transmitting, thus eliminating β. However, unlike TDM, each vehicle does not get exclusive use of the entire channel. The bandwidth must be divided into n slots, and to compensate for effects such as doppler shifts, a buffer, b, must be added between slots. So the bandwidth available to each vehicle is actually

$$B_i = \frac{B - b(n-1)}{n} \tag{5.7}$$

As explained in Chapter 2, the size of b is determined primarily by doppler shifts. A typical size for b is 80 Hz [22].

The maximum utilization is then

$$U = \frac{\sum_{i=1}^{n} B_i}{B} \cdot \frac{T}{S} = \frac{B - b(n-1)}{B} \cdot \frac{T}{S} \tag{5.8}$$

The queueing delay is

$$W = \frac{\lambda n}{2(1-\lambda)} \tag{5.9}$$

time slots [10]. However, since each vehicle has a dedicated frequency channel, propagation delay is not a factor and the duration of a time slot is $S = \alpha + T_i$. The price for a dedicated channel is increased transmission time because the maximum bandwidth available to each vehicle is only B_i. So

$$T_i = \frac{TB}{B_i} \tag{5.10}$$

and

$$S = \alpha + \frac{TBn}{B - b(n-1)} \tag{5.11}$$

As with TDM, the delay increases approximately linearly with n, while the bandwidth per vehicle has an inverse relationship. Further analysis [10] shows that the queueing delay for TDM is $W_{TDM} = W_{FDM} + \frac{n}{2}$, meaning the average wait to transmit data with TDM is longer, especially if n is small or λ is small (sparse data). However if λ approaches 1 and n is large, the transmission time for FDM becomes longer and messages actually get sent with more total delay than with TDM.

5.4.2 Slotted Aloha

Slotted Aloha was developed as an alternative to the multiplexing methods described above. Multiplexing guarantees that there will be no collisions between vehicle transmissions at the cost of bandwidth. For large numbers of vehicles, this cost can be quite high and so multiplexing does not scale well with n. In slotted Aloha, each vehicle transmits its information as soon as it is available. If there is a collision, the involved vehicles wait a random amount of time before retransmitting (Figure 5-6). Because the vehicles do not have to wait for their slot to come up or transmit within a limited frequency space, Aloha

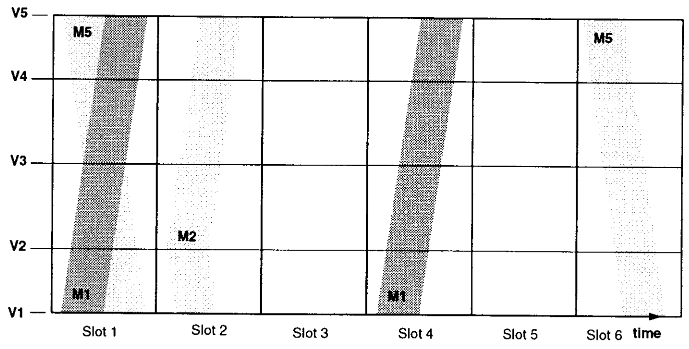


Figure 5-6: Slotted Aloha. Time is broken into slots as with TDM. There is no management and vehicles transmit whenever they have data. In the first slot, vehicle 1 and vehicle 5 transmit, resulting in a collision. Both wait a random number of slots before transmitting again. In the second slot, vehicle 2 is the only one to transmit and all vehicles receive its data. In the fourth slot, vehicle 1 has randomly waited two slots and retransmits its previous message, this time without a collision. In the sixth slot, vehicle 5 retransmits its message.

allows information to be transmitted sooner and more quickly at the cost of a few collisions. For an ideal case, Aloha has a throughput of $\frac{1}{e} \approx 0.368$ timeslots. That is, at its optimum transmission point, approximately 36.8% of Aloha's timeslots contain data. Another 36.8% are empty, while the remainder contain collisions. This optimum point occurs when λ, the rate at which data is queued, equals the rate it is transmitted. If λ increases beyond this point, the frequency of collisions becomes high enough that packets backlog faster than they can be sent out.

By sending data only at the beginning of time slots (hence slotted), we reduce the probability of a collision. Figure 5-6 shows that for slotted Aloha, a time slot only need extend slightly past the propagation delay. Without the slots, we would have to account for a worst-case collision (Figure 5-7) where twice the propagation delay must pass before we detect a collision. So for slotted Aloha, a typical slot length is

$$S = \alpha + T + \beta = \alpha + T + \frac{R}{c} \tag{5.12}$$

Slotted Aloha is useful because this transmission rate of $\frac{1}{e}$ is attained for large n. As shown in Figure 5-9, below the $\frac{1}{e}$ point, Aloha operates much more efficiently than either TDM or FDM with even moderately sized n. At its peak transmission rate, Aloha's utilization is

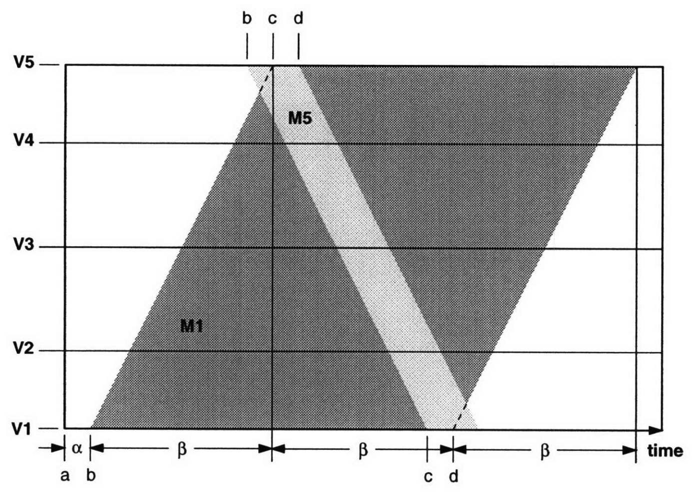


Figure 5-7: Worst-case transmission collision. In this unslotted example, the two furthest vehicles transmit messages resulting in a collision. Because the transmissions are unslotted, vehicle 1 is not aware of the collision until 2β, two propagation delay periods, after it begins transmitting. Compare to slotted transmissions (Figure 5-6) where one propagation delay suffices.

$$U = \frac{T}{S} \cdot \frac{1}{e} = \frac{T}{\alpha + T + \frac{R}{c}} \cdot \frac{1}{e} \tag{5.13}$$

of which is $\frac{1}{e}$ of fully saturated TDM. The price for this high utilization for any n is in collisions. Garbled data packets make up $1 - \frac{2}{e}$ of Aloha's transmissions. These packets must be retransmitted, making the queueing delay more difficult to analyze. For an idealized model such as ours, the queueing delay is

$$W = \frac{e - 1/2}{1 - \lambda e} - \frac{(e^{\lambda} - 1)(e - 1)}{\lambda[1 - (e - 1)(e^{\lambda} - 1)]} \tag{5.14}$$

time slots [10].

5.4.3 Reservations

The previous communications management systems were wasteful at very low and very high data rates. Entire time slots were spent idle or transmitting data which was corrupted upon collision. This can be especially costly if there are large propagation delays as with underwater acoustic communications. A simple solution is for the vehicles to reserve large time slots during which they have exclusive rights to transmit. Because these reservation messages are short, they do not take much time or bandwidth, and there is a lower prob-

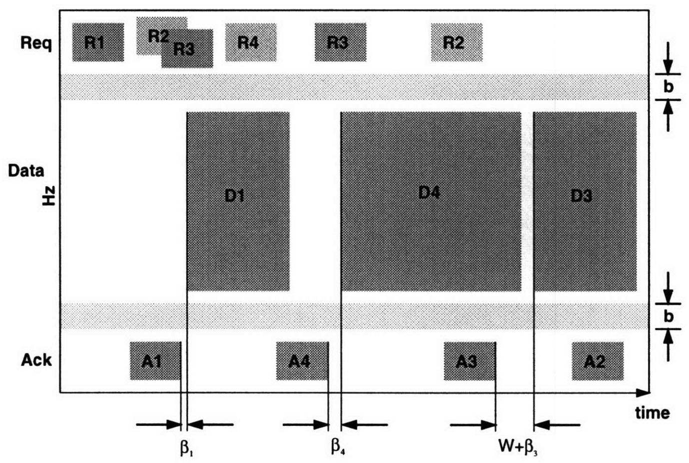


Figure 5-8: A centrally managed reservation system. Frequency space is dived into three regions, one for nodes to make transmission reservation requests, one for the master node to acknowledge reservations, and one for transmitting data. These regions are separated by a buffer, b. Proceeding through time, node 1 requests to send its data (R1). After a propagation delay, the master node gives permission to node 1 (A1). Meanwhile, nodes 2 and 3 send requests (R2 and R3), resulting in a collision. Node 1 has begin transmitting its data (D1). Because of the collision, the master node does not acknowledge either R2 or R3, and the next acknowledgement it sends is for R4. After waiting a random amount of time without an acknowledgement, node 3 retransmits its request. Node 4 begins transmitting its data (D4), during which the master node receives and acknowledges node 3's request, timing its acknowledgement so node 3 will begin transmitting as soon as D4 finishes. Node 2 has also waited a random amount of time and retransmits its request. The master node gives permission for node 2 to transmit its data while D3 is in progress [11].

ability of collision (no chance of collision if using multiplexing). The cost is an increased delay between when the vehicle queues the data and when it is actually sent. The data must wait in the queue while the reservations messages are sent and processed. Two strategies for reservation are presented.

Centralized management

In a centrally managed reservation system, one vehicle or an independent node acts as a master node which processes the reservation requests and assigns transmission times (Figure 5-8). This scheme was proposed by Brady and Catipovic for acoustic modem transmissions in the Monterey Acoustic Local Area Network (ALAN) [11].

In the Brady-Catipovic management scheme, available bandwidth is divided into three frequency spaces, one for requesting reservations, one for acknowledging requests, and another for transmitting data. When a node needs to transmit data, it sends a reservation request to the master node. The master node processes the incoming requests and gives permission to one node at a time to transmit its data. If a collision occurs during a request, the master node does nothing. The nodes sending the request would resend after a random interval if they do not receive an acknowledgement. Their proposal is for acoustic modems fixed on the ocean bottom and a master node at the surface, so the master node can estimate the travel times for acknowledgements to the modems. In this way, incoming data streams from different modems can be timed to arrive almost immediately after the previous transmission ends.

In this centralized management scheme, the request channel operates as unslotted Aloha (Aloha without regular time intervals to denote when transmissions can begin). In ideal unslotted Aloha, $\frac{1}{2e}$ of the transmitted packets contain ungarbled data. Because the request packets are short and infrequent, this limit is usually not reached and collisions are much rarer than if the modems had been transmitting data instead of requests. The reservations allow data transmissions to follow immediately after one another with no collisions; the data channel is almost fully utilized. On the downside, the need for a separate request and acknowledgement channel reduces the amount of bandwidth available to the data channel. It also results in larger delays between when data is ready and when it is finally received. At a minimum, it adds at least the time for signals to reach from the modem, to the master, and back to the modem before data is transmitted.

Decentralized management

The Monterey ALAN is a multipoint-to-point multiaccess channel — the master node is the intended recipient of all data. Hence it makes sense for the master node to arbitrate reservations even though it is on the surface and the nodes are on the ocean floor. In a multiple AUV environment, where the AUVs wish to exchange information among each other, a centralized management scheme would require one vehicle to act as a master. This is undesirable from a robustness standpoint. Should the master vehicle become disabled, the remaining vehicles would be unable to communicate to even decide among themselves which vehicle should become the new master. Also, because the vehicles need to communicate among each other and not only to a master node, much of the advantage of having the master schedule transmissions consecutively is lost. Certainly, a centralized management scheme using a master vehicle could be used. However, for the reasons outlined above and because AUVs will operate in a multipoint-to-multipoint environment, a decentralized reservation management scheme is desirable.

In a decentralized system, reservations are made just as with centralized management. However, the reservations must contain enough information for every vehicle to determine the order of transmission. Whether the reservations are made using TDM, FDM, or Aloha should depend on the circumstances of the deployment. Using the results presented in this chapter and based on number of vehicles, frequency and size of messages, size of operating area, vehicle spacing, etc., the most effective scheme should be selected. The management method for the data channel should also be determined using similar considerations.

In practicality, the simplest method may be FDM for requests and TDM for data. Acoustic modems are usually not able to receive data at the same time they are transmitting, and outfitting a vehicle with more than one modem can be expensive. This would seem to make scheduling reservations independently of data impossible. However, another acoustic source aboard the Odyssey II is the long baseline (LBL) navigation beacon. If vehicles were assigned different LBL frequencies as in FDM, they could use these for the reservation requests. During the reservation period, vehicles with data to transmit would ping, several times to reduce the likelihood of erroneous reception by other vehicles. The order of transmission could be predetermined, perhaps based on vehicle ID number. Then data transmission would then occur using optimized TDM, where a vehicle begins transmitting as soon as it detects the previous vehicle has finished. While the data is being transmitted, a second reservation cycle would be proceeding.

In such a system, our measurement criteria are identical to optimized TDM except for the queueing delay which is shorter because there are no idle transmit cycles. Once a data packet becomes available, it makes a request during the reservation period which occurs concurrently with a data transmit cycle. The data is sent during the next transmit period resulting in a queuing delay of

$$W = \begin{cases} \frac{\lambda\overline{X^2}}{(1-\lambda)} \\ \frac{\lambda\overline{X^2}}{2(1-\lambda)} + \beta & \text{whichever is larger} \end{cases} \tag{5.15}$$

where $\overline{X^2}$ is the mean square of the optimized "time slot" duration [10].

Table 5.1: Summary of multiaccess communication methods.

Method	Time slot (S)	Utilization	Collision	Queueing delay (W) in S
TDM	$\alpha + T + \frac{R}{c}$	$\frac{T}{S}$	0	$\frac{n}{2(1-\lambda)}$
Optimized TDM	$\alpha + T + \frac{R_{i,i+1}}{c}$	$\frac{T}{S}$	0	$\frac{n}{2(1-\lambda)}$
FDM	$\alpha + \frac{TBn}{B-b(n-1)}$	$\frac{B-b(n-1)}{B} \cdot \frac{T}{S}$	0	$\frac{\lambda n}{2(1-\lambda)}$
Slotted Aloha	$\alpha + T + \frac{R}{c}$	$\frac{T}{Se}$	$1 - \frac{2}{e}$	$\frac{e-1/2}{1-\lambda e} - \frac{(e^{\lambda}-1)(e-1)}{\lambda[1-(e-1)(e^{\lambda}-1)]}$
Proposed Res.	$\alpha + T + \frac{R_{i,i+1}}{c}$	$\frac{T}{S}$	0	$2\frac{R_{i,i+1}}{c} + \frac{\lambda \overline{X^2}}{2(1-\lambda)}$

5.4.4 Summary of communications schemes

Table 5.1 summarizes our evaluation criteria for the different communications schemes. Of particular interest is Figure 5-9 which shows the expect queueing delay for the three major schemes discussed, with various numbers of vehicles. As can be seen, slotted Aloha provides the quickest response for low message arrival rates. TDM performs better for moderate to higher arrival rates, but scales linearly with n, making for abyssmal low arrival rate performance. FDM appears to perform best but we should note that this figure gives W based on slot length S. For FDM, the slot length is just the time needed to transmit a message; but because the bandwidth is split among vehicles, this can take much longer than with the TDM or Aloha.

5.5 Recommendations

When designing a multiple-AUV communications system, we should look carefully at the vehicles and the type of deployment. Factors such on the number of vehicles, size of the deployment area, expected average distance between vehicles, and frequency and size of messages are crucial to selecting an optimal communications scheme. For short range operations with few vehicles, or infrequent or short messages, one of the simpler systems such as TDM or FDM may well be the best choice. However, with a large number of vehicles, reservations may be required and collisions may need to be accepted and designed for.

The centralized reservation system developed by Catipovic and Brady works well in a multipoint-to-point environment, where there is a master node to direct all communications. Its distinguishing feature is that the master node is the only node which needs to receive transmissions. If the environment is multipoint-to-multipoint, and all nodes are to receive all signals as in a multi-vehicle environment, and robustness is a concern, then a decentralized

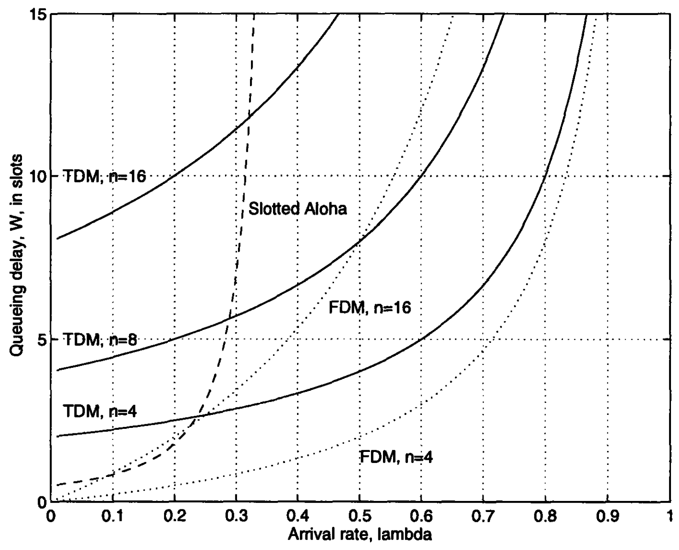


Figure 5-9: Queueing delays for various multiaccess communications methods. W is given in time slots, or the time needed to prepare a signal, transmit it, and wait for it to propagate. With the addition of more vehicles (n), W increases linearly for FDM and TDM. For slotted Aloha, the curve remains the same for all n, but reaches a limit at $\lambda = \frac{1}{e}$. Note that although FDM appears to do the best, the length of TDM and Aloha slots are constant, while FDM slots increase roughly linearly with n. This is due to the FDM bandwidth being divided up between vehicles, causing transmission times to correspondingly be longer (see Table 5.1). In extreme cases with low λ, and short message lengths T or few vehicles n, FDM will perform better than the other methods.

reservation management system is called for.

The main problem with a decentralized system is coding into the request packets what order the vehicles transmit in, and assuring the packet is receive by all vehicles. Up to this point, we had not considered a factor which is mostly inconsequential with the previously described communications schemes. A transmission may not reach all the vehicles. Any time two or more vehicles must communicate, loss of transmission is an unavoidable risk without no guaranteed solution [10]. The only safeguards are to reduce the probability of loss to extremely low levels, and to make the vehicle software robust enough to work with missing data. This usually involves coding error detection and forward error correction into the data at the cost of bandwidth. Also, transmissions can occur more frequently that needed, or vehicles can be programmed to interpolate between missing data.

However, if such a reception error were to occur during a reservation packet, then the affected vehicle would have an incorrect schedule in which data is to be sent. In some cases, if the schedule is garbled just prior to the vehicle's requested transmission time, it may begin transmitting during another vehicle's transmit period.

Another concern is the additional time needed to make a reservation. While reservations can significantly improve transmission rates, their large time delays may cause problems if the information to be transmitted is of critical importance. In addition, one of the simpler communications schemes may be more efficient under a specific set of conditions.

The use of slotted communications brings up the problem of synchronization. For slots to work correctly, all vehicles must know when the slots begin. Keeping the clocks of all these vehicles synchronized is a problem unto itself. For short duration missions, the onboard clock may be sufficient to maintain synchronization. For long deployments, a substantial clock drift may develop, requiring all vehicles be resynchronized.

Finally, the behavior of the acoustic channel was idealized in this analysis. As previously detailed, multipath and other distortions may create shadow zones where transmissions from certain areas will not reach. To further compound the problem, these areas are usually not symmetric - the vehicle which cannot be heard from can hear other vehicles' transmissions just fine. The solution may well lie in modeling the acoustic channel so as to steer vehicles away from these regions, or supress transmissions while traveling in these regions.

5.6 Summary

Unlike communications with a single vehicle, multiple AUV communications is a complex problem of allocating and assigning bandwidth to prevent transmission collisions. TDM and FDM, two methods which eliminate the possibility of collisions, suffer from reduced bandwidth and poor scaling as n, the number of vehicles, increases. Slotted Aloha scales better with n, but at the cost of some collisions. By allowing the vehicles to use some of the available bandwidth for reservations, TDM can be made to scale better with n at the cost of increased complexity. However, the problem is not insurmountable, and the communications method should be chosen after careful consideration of the parameters of the vehicle deployment. Within certain limits, a properly designed multiple AUV communications system will allow vehicles to send telemetry and receive commands with only modest delays and chance of error. In this way, users (and vehicles) can monitor and control each AUV in a multi-vehicle deployment.

Chapter 6

Conclusions

6.1 Overview

Our experiments in the Charles River and Buzzard's Bay have led us to several observations about monitoring and controlling AUVs over serial and network links. We have seen that an unreliable communications channel can cause commands to have the opposite of the intended effect simply due to time delays. Non-uniformity of the modems' coverage area led to additional difficulties obtaining reliable data to base command decisions on, and problems transmitting commands in a timely fashion. A great deal of vehicle functionality can be attained via relatively simple commands, especially with a highly abstracted command behavior list like that on the Odyssey II. The RF modems proved difficult to configure and work with, but ended up unintentionally duplicating some of the reliability problems we expect to encounter with acoustic modems.

The main concern with Internet communications was to reduce bandwidth for multi-vehicle, multi-user communications. To address this issue, we developed our software around the MBone, which introduced other difficulties. Because the MBone is not available with most default network configurations, additional work and networking knowledge is required to set it up properly and track down problems. Because of our findings regarding control signal bandwidth, the need for multicasting is questionable, especially in light of the bandwidth limitations of multi-vehicle communications. The ramifications of multi-vehicle communications on multi-user Internet access need to be futher evaluated.

We chose Matlab for our user interface to maintain compatibility across different computer platforms. However, we experienced several problems with the display portion of the

GUI using all available computer resources and making the control portion unresponsive. In addition, because the user lacked visual feedback of the actual vehicle and the communications channel was known to be unreliable, confidence in the GUI's display was low and commands to the vehicle were conservative. This was aggravated by the GUI relying on the vehicle's dead-reckoned track, rather than positioning based on a fixed navigational array.

6.2 Lessons Learned

From our experiments, we have determined several key issues which must be addressed for a control and monitoring communications link to work successfully.

6.2.1 AUV operations

Foremost, the AUV must assume that each command given to it may be the last it receives. Since the communications channel is unreliable, the data stream may become corrupted, truncated, or lost entirely. In each of these situations, the AUV must be able to recover to a state in which it is still functional and commandable. Corrupted and truncated data can be screened out or recovered through error checking and correcting schemes. In the case of lost commands, the vehicle's onboard intelligence must be able to recognize when it may be entering an unrecoverable state (leaving the communications area of coverage or destruction of the vehicle), and autonomously override its current command.

The area of communications coverage is non-uniform. In our experiments with an RF modem, we used a directional antenna which offered greater range when directly in line with the vehicle. This resulted in a lobe-shaped area of coverage whose borders were difficult to determine and define geometrically. With acoustic communications, there may be shadow zones where control signals cannot reach the vehicle, and the signal strength will vary depending on the sound speed profile and other acoustic factors. To prevent the vehicle from drifting outside communications range, a behavior such as a bounding box limiting the vehicle's area of operation should be used. Defining the limits of this "box" is an involving problem in itself, requiring modeling of the acoustic channel.

Time delays in transmitting, processing, and implementing commands may cause unexpected or undesired behaviors. While not as critical as keeping the vehicle out of unrecoverable states, a control interface which allows the operator to look ahead into the vehicle's projected path at the time the command is implemented is desirable, and would prevent nasty surprises.

When controlling the vehicle, reliability is more important than bandwidth. Short, simple commands are capable of controlling the vehicle heading and depth, providing a "lowest common denominator" set of commands which require very little bandwidth. However, if these short commands cannot reach the vehicle reliably, all control is lost.

The RF link proved tempermental with reliability problems which were unreproducible. Packets would be dropped without any discernible pattern, the modems would buffer and try to resend these dropped packets even when commanded not to do so, and every few runs the modems would lose contact with each other, requiring us to power cycle them to reinitialize the link. Near one of our final experiments we determined that water getting into the antenna was likely the cause of most of these failures. The final two runs with the antenna mounted inside the box produced constant data without any dropouts. However, without further testing we cannot state conclusively that the antenna caused all our problems. We experienced more problems than we would be comfortable with in a critical application in which daily use was vital for success of the experiment.

6.2.2 Internet communications

In general, the Internet communications portion of our experiments worked well — more reliably than the RF portion. We had no problems with dropouts or buffering delays as we did with the RF communications. However, there were several Internet-related issues which arose during our experiments.

Because the Internet is based on point-to-point communications, it is not well designed for multi-node problems. For n nodes to communicate with each other, the number of required point-to-point links increases polynomially in n. For one or a few nodes, this may be acceptable. However, as the number of nodes increases, multicasting (point-to-multipoint communication) becomes more desirable. However, we have found that very little bandwidth is required to control a single vehicle, and the multi-vehicle problem calls for few, short messages, and so the need to multicast all communications becomes questionable. Work should be done to investigate how the communications problem scales with additional vehicles and users.

The ability to multicast over the Internet is a work-around to the design of the Internet.

It works by overlaying multicast features on top of existing Internet links. Because of this and its lack of wide availability, it is not as robust as standard Internet links and requires more maintenance work by the user. For example, MIT network services shut down our multicast repeater before an experiment, and it took us several hours to diagnose the problem because other Internet services were working properly. In the case of a remote deployment where the only Internet link may be through a generic Internet service provider, considerable work may be required to establish a connection to the multicast backbone.

One of our design criteria was that any Internet communications software developed should be platform independent. Unfortunately, at the onset of this research, it was difficult to produce and maintain network code that would work with Unix, Macintosh, and PC compatible computers. The recent growth of the World Wide Web (WWW) and introduction of Java has changed this. The Virtual Reality Modeling Language (VRML), a 3-D extension to the WWW, shows promise as a means of displaying AOSN data via a web browser on any platform, using the WWW as the medium for network data transmission. The wide acceptance of Java as a web browser extension on nearly all computer platforms means the same network source code could run on all these systems without modification. However, since the WWW is based on TCP/IP, both VRML and Java suffer from the same bandwidth problems mentioned previously. Future work will have to consider the benefits of cross-platform compatibility offered by the WWW against the improved network performance of the MBone and multicast routines.

6.2.3 User Interface

There is a psychological feedback factor involved with control over the internet. Without a history of successful operation, and without being able to watch the vehicle as it moves, the remote user's confidence in the vehicle's location is low. For our experiments, we always had a person (with override control) monitoring the mission from the river, which defeats the purpose the controlling the vehicle remotely. This problem should lessen as we gain more experience and build up a history of successful operation.

Dead reckoning the vehicle's position is insufficient for all but the simplest of missions and tests, especially in light of the previous problem. Accumulated drift and uncertainty in the vehicle's velocity reduce confidence in the vehicle's position, which reduces the effectiveness of control, and renders some of the previously mentioned self-preservation behaviors useless. A more robust, verifiable navigation scheme is desirable.

On occasion, our control software would freeze or lock up for several seconds. While these incidents were annoying, and in some critical instances heart-stopping, they were attributable to our software implementation, not to our communications links. Furthermore, with the proper safeguards described above in place, such freezes should appear no different to the vehicle than a communications dropout, and the vehicle would not be jeopardized.

The original intent was to use Matlab as a cross-platform display and control interface, with a different external interface routine for each computer platform containing platform-specific network code. However, because the display portion of Matlab scripts interfered with the control portion, the interface seemed unresponsive to the user. Future implementations can address these concerns with a dedicated graphics interface utilizing cross-platform application libraries, such as OpenGL, X, or Tcl.

6.3 Multiple AUV Communications

The characteristics of several multiaccess communications methods can be compared quantitatively to decide which method is best suited for a particular mission. For very short, infrequent messages and few vehicles, frequency domain multiplexing provides collision-free communications. For short messages in a small operating area, time domain multiplexing allows messages to be sent more quickly than FDM, at the cost of longer waits before a message is actually sent. If messages occur very frequently, the queueing delay for both FDM and TDM approach the same value.

The queueing delay for FDM and TDM increase linearly with the number of vehicles, n, making these methods impractical for large numbers of vehicles. One way to overcome this problem is with slotted Aloha, where vehicles transmit their data as it becomes available. Aloha has cannot handle as many messages as FDM or TDM, but for low message rates, it does not degrade as n increases. Theoretically, almost an infinite number of vehicles can use Aloha as long as the overal message rate does not exceed $\frac{1}{e}$ of the available time slots.

Reservations work well in enhancing multiaccess communications for environments with large propagation delays. Using a reservation system, TDM can continue to work well for large numbers of vehicles without scaling linearly in n. However, deciding how to make reservations and guaranteeing all vehicles simultaneously decide upon the same reservation order is a difficult problem.

6.4 Future Work

Our research suggests several avenues of future research. These range from addressing the problems discussed above, to extending the research into other areas not touched on by this work.

Behaviors need to be designed to prevent the vehicle from entering an unrecoverable state due to unreliable communications or a catastrophic event such as hitting a wall. Our circling behavior represents only a simple, partial solution to this problem. These behaviors could incorporate boundaries based on the communications coverage area, and known or newly discovered obstacles in the vehicle's path. Or a behavior could cause the vehicle to return towards its launch point if a command has not been received in a set amount of time.

Towards this end, modeling of the acoustic channel in which the vehicle will operate is of major importance. The bending of acoustic signals can lead to asymmetric shadow zones in which commands cannot reach the vehicle, telemetry cannot reach the user, or both. Multipath can cause acoustic signals to add or cancel at certain locations. By modeling these types of phenomena prior to a mission, the vehicle can be programmed (or guided) around or through these points with minimal inconvenience and surprise to the user.

Behaviors based on fixed boundaries become increasingly inaccurate with time if the vehicle position is not based on a fixed navigational array. Because our results are based on dead-reckoned position, the lack of user confidence in the GUI display may not be a problem with a navigational array. On the other hand, inaccurate navigation fixes due to echoes and garbled navigation beacon signals may cause this problem to become even worse. The interaction of different positioning system with the usability of vehicle telemetry data needs to be further studied.

Regrettably, an acoustic modem pair was unavailable for these experiments. Since the bandwidth limitations and expected reliability were based on real acoustic modems, the results of these radio modem experiments should be transferable to acoustic modems. However, experiments with real acoustic modems may reveal problems or solutions not readily apparent from these experiments.

A rudimentary control language based on the Odyssey II's data structure was developed in this work. This language should be further expanded, and possibly genericized for use with any AUV and not just the Odyssey II. This would open up the possibility of using the same monitoring and control software with any AUV connected to a network such as AOSN. This language could also be modified or merged with other simulation protocols such as DIS [34], so monitoring and control software developed for those applications could be used with AUVs.

This thesis only experimented with communications between one vehicle and one user. While provisions were made for incorporation of multiple users and vehicles in the future, and theoretical work was done towards this goal, no multiple AUV experiments were conducted. Conclusions regarding the best way to incorporate multi-user capability remain vague, especially with the rapid development of Internet technologies. The problem of bandwidth reduction due to multiple vehicles sharing the communications channel has solutions, but it remains to be seen whether these solutions are practical for real world applications.

Finally, the tradeoff between communications reliability and bandwidth should be further investigated. Recent experiments at Haro Strait [44] indicate that a bare minimum of communications bandwidth is sufficient to usefully control the vehicle, so long as the communications channel is reliable. Whether this is also true for data telemetry remains to be seen and should be researched further.

6.5 Summary

Several conclusions can be drawn from our experiments about monitoring and controlling AUVs remotely. A great deal of control of the vehicle can be attained with relatively short and simple commands. But an unreliable communications channel can lead to all sorts of problems. Perhaps AI behaviors can be implemented to work around the problems. Because the coverage by the modems is neither infinite nor uniform, AUV operations can be facilitated if weak spots in coverage can be pinpointed beforehand. Use of multicasting routines solve the problem of scaling to multiple users, but introduce other problems with the usability of the system as a whole. While Matlab seemed a good choice for a cross-platform user interface, there were problems inherent in Matlab's sharing of computer resources, and other cross-platform user interfaces should be investigated. In time, it should be possible to monitor and control several AUVs from an ordinary WWW browser such as Netscape.

Appendix A

Program listings

This appendix lists the key computer programs developed in the course of this research. The routines are divided into three categories: Matlab scripts and CMEX code used for monitoring and controlling the vehicle, C code additions and modifications to the software aboard the vehicle, and C code written to run on the SGI workstations.

A.1 Matlab Scripts

A.1.1 gr_control.m

```
% gr_control.m
% clears all variables and calls the two components of the control software:
% gui.m for controlling the vehicle
% ody_view.m for monitoring the vehicle

clear
gui
ody_view
```

A.1.2 ody_view.m

```
% ody_view.m
% generates dead reckoned position and depth plots which are updated in
% real time using the expandwin.m routine

if ( exist( 'xyplot' ) ~= 1 )
  xyplot = figure;
  set( xyplot, 'position',[12 349 564 409] );
  set( xyplot, 'Name', 'Vehicle Trajectory' );
end
```

```
if ( exist( 'zplot' ) ~= 1 )
  zplot = figure;
  set( zplot, 'position',[12 13 564 300] );
  set( zplot, 'Name', 'Depth and calculated bottom' );
end

if ( exist( 'duration' ) ~= 1 ), duration = 1800; end

time = 0;
east = 0;
nrth = 0;
dr_east = 0;
dr_nrth = 0;
dpth = nan;
botm = nan;
odydat=recv_data(0); % initialize recv_data multicast routines

while isempty( odydat )
  odydat=recv_data(1); % receive multicast data
end
time = odydat(1); % time
st_time = time;
dr_nrth = odydat(6);
dr_east = odydat(7);
east = dr_east;
nrth = dr_nrth;

figure( xyplot ); clf;
htrack = plot( dr_east, dr_nrth, 'r+' );
title( 'Odyssey Track by Dead Reckoning', 'erasemode', 'none' );
xlabel( 'East (m)', 'erasemode', 'none' );
ylabel( 'North (m)' );
set( htrack, 'erasemode', 'none' );

set( gca, 'aspectratio', [1 1], 'drawmode', 'fast', ...
  'xgrid', 'on', 'ygrid', 'on' );
click_heading % mouse_click for new heading

figure( zplot ); clf;
hdepth = plot( time, dpth, 'r' );
title( 'Measured Depth and Calculated Bottom', 'erasemode', 'none' );
xlabel( 'time (s)', 'erasemode', 'none' );
ylabel( 'depth (m)' );
set( hdepth, 'erasemode', 'none' );
hfloor = line( time, botm );
set( hfloor, 'erasemode', 'none', 'color', 'b' );
set( gca, 'aspectratio', [2 nan], 'drawmode', 'fast', 'ydir', 'reverse', ...
  'xgrid', 'on', 'ygrid', 'on' );

set( xyplot, 'position',[12 349 564 409] );
set( zplot, 'position',[12 13 564 300] );

j = 2;
begin = clock;
while etime( clock, begin ) < duration
  cyclebegin = clock;
  odydat=recv_data(1); % receive multicast data
  if ( ~isempty( odydat ) )
    cp = cos( odydat(3)*pi/180 );% cosine of pitch
    ds_hor = 1.4*(odydat(1) - time(j-1))*cp; % fudge this to chg speed
    time(j) = odydat(1);% time
    east(j) = east(j-1) + ds_hor*sin( odydat(2) );
    nrth(j) = nrth(j-1) + ds_hor*cos( odydat(2) );
    dpth(j) = odydat(4);% depth
    botm(j) = odydat(4) + odydat(5)*cp;% bottom
    dr_nrth(j) = odydat(6);
    dr_east(j) = odydat(7);
    figure( xyplot );
    set( htrack, 'xdata', dr_east, 'ydata', dr_nrth );
    expandwin( dr_east(j), dr_nrth(j), j );
    figure( zplot );
    set( hdepth, 'xdata', time, 'ydata', dpth );
    set( hfloor, 'xdata', time, 'ydata', botm );
    expandwin( time(j), botm(j), j );
    j = j + 1;
  end
end
duration
figure( xyplot );
set( gca, 'xlimmode', 'auto', 'ylimmode', 'auto' );
figure( zplot );
set( gca, 'xlimmode', 'auto', 'ylimmode', 'auto' );
```

A.1.3 expandwin.m

```
% expandwin.m
% dynamically rescales and redraws a Matlab plot.
% unknown origin.  Matlab homepage?

function limits = expandwin( x, y, flag )

if nargin < 3, flag = 1; end

if ( flag == 1 )
```

```
tick = get( gca, 'xtick' );
limits(1:2) = [tick(1) max(tick)];
tick = get( gca, 'ytick' );
limits(3:4) = [tick(1) max(tick)];
else
limits = axis;
end

change = 0;

if ( x > limits(2) )
wid = ceil(1.1*( x - limits(1) ));
set( gca, 'xlim', [limits(1) limits(1)+wid ] );
change = 1;
end
if ( x < limits(1) )
wid = ceil(1.1*( limits(2) - x ));
set( gca, 'xlim', [limits(2)-wid limits(2)] );
change = 1;
end
if ( y > limits(4) )
hgt = ceil(1.1*( y - limits(3) ));
set( gca, 'ylim', [limits(3) limits(3)+hgt] );
change = 1;
end
if ( y < limits(3) )
hgt = ceil(1.1*( limits(4) - y ));
set( gca, 'ylim', [limits(4)-hgt limits(4)] );
change = 1;
end

drawnow;
if ( change == 1 )
xlim = get( gca, 'xlim' );
ylim = get( gca, 'ylim' );
end
```

A.1.4 recv_data.c

```
/* recv_data.c
 * Opens multicast port, reads incoming data, and converts format to
 * something ody_view will understand.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "../../multicast/mcast.h"
#include "/usr2/auvlab/matlab/extern/include/mex.h"

#define MAX_X 20
char colon[2] = ":\0";

struct value {
  char name[30];
  int index;
};

void mexFunction(
                 int nlhs, Matrix *plhs[],
                 int nrhs, Matrix *prhs[])
{
  double *init;
  double time, pitch, yaw, depth, altitude, dr_north, dr_east;
  int cnt, msglen, cmp;
  int i, j, j_count;
  char message[1500];
  char netaddr[50];
  char *tmp_chr;
  unsigned int chksum;
  char name[MAX_X][50];
  double x[MAX_X];
  static struct value n[MAX_X];

  if (nrhs != 1) {
    mexErrMsgTxt("One input arguments required.");
  } /* eise if (nlhs > 0) {
       mexErrMsgTxt("No ouput arguments required.");
       } */

/*  printf("inside recv_data routine\n"); */

  init = mxGetPr(prhs[0]);

  if (*init == 0) {
    Mcast_init(1); /* Read in multicast configuration and initialize */
    Mcast_set_nonblock(1);
    strcpy( n[0].name, "dummy\0" );
    n[0].index = 0;
    x[0] = 0; /* dummy value */
    for (j=1; j<MAX_X; j++) {
      n[j].index = -1;
      strcpy( n[j].name, "dummy\0" );
    }
    strcpy( n[1].name, "m_present_time\0" );/* Are in the order ody_view    */
    strcpy( n[2].name, "m_pitch\0" );       /* expects them.  Incoming data */
    strcpy( n[3].name, "m_heading\0" );     /* stream can be in any order   */
    strcpy( n[4].name, "m_depth\0" );
    strcpy( n[5].name, "m_sonar_alt\0" );
    strcpy( n[6].name, "m_dr_north\0" );
    strcpy( n[7].name, "m_dr_east\0" );

  } else if (*init > 0) {
    cnt = Mcast_receive(message, sizeof(message), netaddr);
    if (cnt < -1) {
      perror("recvfrom");
      exit(1);
    } else if (cnt > 0) {
      /*      message[cnt]='\0'; */
      fprintf(stderr, "recv_data> %s: message = \"%s\", size:%d\n",
              netaddr, message, cnt);

      /*
       * Look for the string "VALUES>" or "NAMES>" in "message"
       */

      if (( strncmp( message, "VALUES>", 7))==0) {

        tmp_chr = strtok( message, " ");  /* gobble VALUES> */
        tmp_chr = strtok( NULL, " ");
        j_count = -1;
        while (( tmp_chr != NULL) && ( j_count < (MAX_X-1) )) {
          if ( tmp_chr[0] == colon[0] ) { /* colon found, checksum coming up */
            tmp_chr = strtok( NULL, " ");
            sscanf (tmp_chr, "%02X", &chksum);
          } else {                         /* normal value */
            j_count++;
            sscanf (tmp_chr, "%lf", &(x[j_count]));
            tmp_chr = strtok( NULL, " ");
          } /* else */
        } /* while */

        if ( ( j_count == (MAX_X-1)) && ( tmp_chr != NULL ) ) {
          fprintf (stderr, "sgi_comm> Got more than MAX_X values,");
          fprintf( stderr, "ignoring all but first %d.\n",   j);
        }

        fprintf( stderr, "recv_data parse> %d values: ", j_count );
        for ( j=0; j<(j_count); j++)
          fprintf( stderr, "%5.3lf ", x[j] );
        fprintf( stderr, ": %02X", chksum);
        fprintf( stderr, "\n");
  /*        if (j_count == 7) { */
  if ( j_count>1 ) {
    double *out;
    plhs[0] = mxCreateFull( 1, 7, REAL );
    out = mxGetPr( plhs[0] );
    out[0] = x[n[1].index]; /* time */
    out[1] = x[n[2].index]; /* pitch */
    out[2] = x[n[3].index]; /* yaw */
    out[3] = x[n[4].index]; /* depth */
    out[4] = x[n[5].index]; /* altitude */
    out[5] = x[n[6].index]; /* dr_north */
    out[6] = x[n[7].index]; /* dr_east */
    fprintf(stderr,
            "recv_data matlab> t:%g p:%g y:%g d:%g a:%g dn:%g de:%g\n",
            out[0], out[1], out[2], out[3], out[4], out[5], out[6]);
  }
  return;
} else if (( strncmp( message, "NAMES>", 6))==0) {
  /* process list of names */

  tmp_chr = strtok( message, " ");  /* gobble NAMES> */
  tmp_chr = strtok( NULL, " ");
  j_count = -1;
  while (( tmp_chr != NULL) && ( j_count < (MAX_X-1) )) {
    if ( tmp_chr[0] == colon[0] ) { /* colon found, checksum coming up */
      fprintf( stderr, "looking for checksum\n");
      tmp_chr = strtok( NULL, " ");
      sscanf (tmp_chr, "%02X", &chksum);
    } else {                    /* normal value */
      j_count++;
      /*            sscanf (tmp_chr, "%s", name[j_count]); */
      strcpy( name[j_count], tmp_chr );
      fprintf( stderr, "name %d is %s\n", j_count, name[j_count]);
      tmp_chr = strtok( NULL, " ");
    } /* else */
  } /* while */

  if ( ( j_count == (MAX_X-1)) && ( tmp_chr != NULL ) ) {
    fprintf (stderr, "sgi_comm> Got more than MAX_X names,");
    fprintf( stderr, "ignoring all but first %d.\n",   j);
  }

  fprintf( stderr, "recv_data parse> %d values: ", j_count );
  for ( j=0; j<(j_count); j++)
    fprintf( stderr, "%s ", name[j] );
  fprintf( stderr, ": %02X", chksum);
  fprintf( stderr, "\n");
```

```
        for ( i=0; i<MAX_X; i++ ) {
          msglen = strlen(n[i].name);
          for (j=0;j<=j_count;j++) {
            cmp = strncmp(n[i].name, name[j], msglen);
            if (cmp==0) {
              fprintf( stderr, "match: name %d %s %d %s\n",
                       i, n[i].name, j, name[j] );
              n[i].index = j;
              j=j_count;
            } /* if (cmp==0) */
          } /* for j */
          if (cmp)
            fprintf( stderr, "unable to match name %d \"%s\"\n",
                     i, n[i].name );
        } /* for i */
      } /* else if NAMES */
    } /* else if cnt */
  } /* else if *init */
}
```

A.1.5 gui.m

```
% gui.m
% Sets up a dialog box for sending new commands to the vehicle

send_sp(-1,0,0,0) % init multicast send

if ( exist( 'gui' ) ~= 1 )
  gui = figure;
end

figure( gui ); clf;
set( gui, 'position', [12 795 290 195] );
set( gui, 'Name', 'User control panel' );


send_setpoint = uicontrol(gcf,...
  'Style','push',...
  'String','Send Setpoint',...
  'Position', [10 10 120 25],...
  'CallBack', [...
    'sp_error = 0;,',...
    'sp_time = str2num(get(en_time,''String'')),',...
      'if sp_time <= 0,',...
        'disp(''ERROR - Value of time must be positive''),',...
        'sp_error = 1;,',...
      'elseif size(sp_time) == 0,',...
        'disp(''ERROR - Time must be a numerical value''),',...
        'sp_error = 1;,',...
      'end,',...
    'sp_speed = str2num(get(en_speed,''String'')),',...
      'if sp_speed <= 0,',...
        'disp(''ERROR - Value of speed must be positive''),',...
        'sp_error = 1;,',...
      'elseif size(sp_speed) == 0,',...
        'disp(''ERROR - Speed must be a numerical value''),',...
        'sp_error = 1;,',...
      'end,',...
    'sp_depth = str2num(get(en_depth,''String'')),',...
      'if sp_depth <= 0,',...
        'disp(''ERROR - Value of depth must be positive''),',...
        'sp_error = 1;,',...
      'elseif size(sp_depth) == 0,',...
        'disp(''ERROR - Depth must be a numerical value''),',...
        'sp_error = 1;,',...
      'end,',...
    'sp_heading = pi*str2num(get(en_heading,''String''))/180,',...
      'if sp_heading < 0,',...
        'disp(''ERROR - Value of heading must be positive''),',...
        'sp_error = 1;,',...
      'elseif sp_heading >= 2*pi,',...
        'disp(''ERROR - Heading must be less than 360 degrees''),',...
        'sp_error = 1;,',...
      'elseif size(sp_heading) == 0,',...
        'disp(''ERROR - Heading must be a numerical value''),',...
        'sp_error = 1;,',...
      'end,',...
    'if sp_error == 1,',...
      'disp(''Error occurred - unable to send setpoint''),',...
    'else,',...
      'send_sp(sp_heading, sp_depth, sp_speed, sp_time),',...
    'end,',...
    'end']);

cancel_setpoint = uicontrol(gcf,...
  'Style','push',...
  'String','Cancel Setpoint',...
  'Position',[160 10 120 25],...
  'CallBack','close(gcf)');

frame_vars = uicontrol(gcf,...
  'Style','frame',...
  'Position',[10 55 150 130]);
txt_vars = uicontrol(gcf,...
  'Style','text',...
  'String','Variables',...
  'Position',[10 160 150 25]);
txt_heading = uicontrol(gcf,...
  'Style','text',...
  'String','Heading (deg)',...
  'Position',[10 135 105 25]);
txt_depth = uicontrol(gcf,...
  'Style','text',...
  'String','Depth (m)',...
  'Position',[10 110 75 25]);
txt_speed = uicontrol(gcf,...
  'Style','text',...
  'String','Speed (m/s)',...
  'Position',[10 85 85 25]);
txt_time = uicontrol(gcf,...
  'Style','text',...
  'String','Time (s)',...
  'Position',[10 60 65 25]);

en_heading = uicontrol(gcf,...
  'String', '170',...
  'Style','edit',...
  'Position',[115 135 40 25]);
en_depth = uicontrol(gcf,...
  'String', '2',...
  'Style','edit',...
  'Position',[115 110 40 25]);
en_speed = uicontrol(gcf,...
  'String', '1',...
  'Style','edit',...
  'Position',[115 85 40 25]);
en_time = uicontrol(gcf,...
  'String', '60',...
  'Style','edit',...
  'Position',[115 60 40 25]);
```

A.1.6 send_sp.c

```
/* send_sp.c
 * Routine for multicasting messages given to it by Matlab
 *
 * icrc is the CRC routine from Numerical Recipes in in C

#include <stdio.h>
#include "../multicast/mcast.h"
#include "/usr2/auvlab/matlab/extern/include/mex.h"

/* #define CRC */
```

```
unsigned short icrc(unsigned short crc, unsigned char *bufptr,
        unsigned long len, short jinit, int jrev);

void send_sp(double *heading, double *depth, double *speed, double *time)
{
  char message[1500];
  char tempstr[1500];
  char netaddr[50];
  int msgsz, cnt;
  unsigned short crc;

  if (*heading == -1) {
      Mcast_init(2); /* Read in multicast configuration and initialize */
    } else {
      sprintf(message, "CMD> setpoint %f %f %f %f",
              *heading, *depth, *speed, *time);

#ifdef CRC
      crc = icrc(0, message, strlen(message), 0, 1);
      fprintf(tempstr, "%s : %d", message, crc);
      strcpy(message, tempstr);
#endif

      fprintf(stderr, "message = %s\n", message);
      msgsz=strlen(message);
      cnt = Mcast_send2(message, msgsz);
      if (cnt < 0) {
        perror("sendto");
        exit(1);
      }
    }
}

/* Gateway Routine */
void mexFunction(
                 int nlhs, Matrix *plhs[],
                 int nrhs, Matrix *prhs[])
{
  double *heading, *depth, *speed, *time;

  if (nrhs != 4) {
    mexErrMsgTxt("Four input arguments required.");
    } /* eise if (nlhs > 0) {
       mexErrMsgTxt("No ouput arguments required.");
       } */

  printf("inside gateway routine\n");
  heading = mxGetPr(prhs[0]);
  depth = mxGetPr(prhs[1]);
  speed = mxGetPr(prhs[2]);
  time = mxGetPr(prhs[3]);

  send_sp(heading,depth,speed,time);
}
```

A.2 Vehicle C Code

A.2.1 comm.c

This routine runs on the vehicle, independent of the main vehicle code. It links directly with the main code's data structure, relaying commands and data between the modem and main code.

```
    ⋮

main(argc, argv)
     int argc;
     char *argv[];
{

⋮

  /* Reads in sensor_names to transmit from file out_array.name*/

⋮

  /* for each sensor_name in out_array.name, check the entire
     list of sensor values to find a match */

⋮

  /* link to vehicle structure date module */

⋮

  while (!mission_finished) {      /* main loop */
    /* set alarm to send signal 259 after n_ticks_to_sleep ticks
       (10ms/tick) */

    if((alarm_id = alm_set(259, n_ticks_to_sleep)) == -1)
      fprintf(stderr, "Can't set the alarm in modem\n");

    t1 = t2;
    t2 = millisec();
```

```
cycle_count++;


if (vvp->abort_mission) {
  fprintf( term,
           "*** comm: exiting because vvp->abort_mission=%d ***\n",
           vvp->abort_mission );
  fprintf( syslog,
           "*** comm: exiting because vvp->abort_mission=%d ***\n",
           vvp->abort_mission );
  exit_gracefully( vvp->abort_mission );
}




/* fprintf(stderr, "comm> %d time %d\n", cycle_count, millisec()); */
fprintf(syslog, "comm %7d>\n", cycle_count, millisec());
/* use a routine called report to write the value and name of
   selected variables that we are interested in.  For now,
   each will have its own line, but maybe in the future we
   can do some nifty formatting or a non-scrolling screen */


/* get input from the user */
cmd[0] = '\0';
n_to_read = _gs_rdy( stdin_port );
if (n_to_read > 0 ) {
  conf = read( stdin_port, cmd, n_to_read );
  if (conf != n_to_read) {
    fprintf( term, "**** Error: attempted to read %d chars"
             "read returns %d\n", n_to_read, conf );
  } else {
    cmd[n_to_read] = '\0';
    chksum = calc_check_sum( cmd);
  }
}

/* if we received a command, parse it and try to follow it */
if (cmd[0] != '\0') {
  conf = parse_command( cmd );
  if (conf==-1)  {
    fprintf( term, "\nbad cmd> %s\n", cmd );
    fprintf( syslog, "\nbad cmd> %s\n", cmd );
  }
}

/* simply write the command into the "modem_rec" array
   of the vehicle data structure */
```

```
    if (strlen(cmd) < 500) {
      conf_ptr = strcpy( vvp->modem_rec, cmd);
      if (conf_ptr==NULL)  {
        fprintf( term, "\nerror writing cmd to vvp->modem_rec\n");
        fprintf( syslog, "\nerror writing cmd to vvp->modem_rec\n");
      }
    }

    report( out_index, num_outputs);

    /* fprintf(syslog,
       "comm> going to sleep, elapsed time for cycle: %d\n",
       millisec() - t2); */
    j = sleep(1); /* this sleep should never finish */

    alm_delete(alarm_id);  /* delete alarm */

    if (num_cycles)
      if (cycle_count==num_cycles) mission_finished=1;
    if (vvp->comm_synchro==-1) {
      mission_finished=1;
      fprintf(stderr, "comm> comm_synchro=-1, exiting\n");
    }

  }

  exit_gracefully(0);
}
⋮

int report(int *out_ind, int num_out )
{
  int i, msg_len=0;
  int uniform_format = 1;

  static int print_names = 1;
  char msg[500] = "", buf[50];
  unsigned char chksum=0;


  /* the check sum is everything up to but not including the colon */
  if (print_names) {
    print_names=0;
    sprintf(msg, "NAMES> ");
    for (i=0;i<num_out;i++) {
      strcat(msg, vvp->s[out_ind[i]].name);
      strcat(msg, space);
    }
    /* attach checksum and then cr_lf */
    chksum = calc_check_sum( msg );
    strcat(msg, colon);
    strcat(msg, space);
    sprintf( buf, "%02X", chksum);
    strcat(msg, buf);
    strcat(msg, crlf);
    strcat(msg, space);
    fprintf( term, "%s", msg);
  }

  sprintf(msg, "VALUES> ");
  for (i=0;i<num_out;i++) {
    if (uniform_format)
      sprintf( buf, "%6.3lf ", vvp->s[out_ind[i]].x);
    else
      sprintf( buf, "%.5g ", vvp->s[out_ind[i]].x);
    strcat( msg, buf);
  }
  chksum = calc_check_sum( msg );
  strcat(msg, colon);
  strcat(msg, space);
  sprintf( buf, "%02X", chksum);
  strcat(msg, buf);
  strcat(msg, crlf);
  fprintf( term, "%s", msg);
}

unsigned char calc_check_sum( char *buf )
{
  int buf_len, i;
  unsigned char chk_sum;

  buf_len = strlen(buf);
  chk_sum = 0;
  for (i=0;i<buf_len;i++)  {
    chk_sum += buf[i];
  }
  return chk_sum;
}

⋮

int parse_command( char *cmd)
{
  int count=0, conf, i, setpoint_priority=-1, debug=0;
```

```
char *tmp_chr;            /* not sure why needs to be ** */
float arg[10];

if (debug)
  fprintf( term, "\nCalling parse_command()\n");

/* use strtok to parse the command, which is a sequence of
   words separated by spaces */

/* first, gobble up the CMD> */
tmp_chr = strtok( cmd, " ");
if (debug) fprintf( term, "parse_command() ... called strtok\n");

if (tmp_chr==NULL) return -1;
conf=strcmp(tmp_chr, "CMD>");
if (debug)
  fprintf( term, "strcmp(tmp_chr, \"CMD>\") returns %d\n", conf);
if (conf) return -1;


/* get next word and see if it is "abort" */
tmp_chr = strtok( NULL, " ");
if (tmp_chr==NULL) return count;
fprintf( syslog, "command word %d |%s|\n", count, tmp_chr);
if (debug) fprintf( term, "command word %d |%s|\n", count, tmp_chr);

conf=strcmp( tmp_chr, "abort");
if (conf==0) {
  if (debug) fprintf( term, "*** command match: ABORT\n");
  fprintf( syslog, "*** command match: ABORT\n");
  vvp->abort_mission = 1;
}

/* setpoint assumes you give four arguments: heading, depth, speed, time */
conf = strcmp( tmp_chr, "setpoint");
if (conf==0) {
  if (debug) fprintf( term, "*** command match: setpoint\n");
  fprintf( syslog, "*** command match: setpoint\n");
  for (i=0;i<4;i++) {
    tmp_chr = strtok( NULL, " ");
    if (tmp_chr==NULL) {
      fprintf( term, "cmd setpoint ignored: only %d args\n", count);
      return count;
    }
    count++;
    if (debug)
      fprintf( term, "command word %d |%s|\n", count, tmp_chr);
    conf = sscanf(tmp_chr, "%f", &(arg[i]));
    if (conf!=1) {
      fprintf( term, "error parsing setpoint argument %s, sscanf "
               "returns %d\n", tmp_chr, conf);
    }
  }
  /* assuming all four arguments were given, write the values
     to the setpoint behavior with the highest priority
     (lowest on stack) */
  for (i=0;i<MAX_COMMAND;i++) {
    fprintf( stderr, "testing match with beh[%d] = %s ptr 0x%X\n",i,
             vvp->sta[vvp->cstate].c[i].b.name,
             (int) vvp->sta[vvp->cstate].c[i].b.fcn);

    if (strncmp(vvp->sta[vvp->cstate].c[i].b.name, "setpoint", 8)
        ==0) {
      fprintf( stderr, "match for behavior %d %s\n", i,
               vvp->sta[vvp->cstate].c[i].b.name);

      setpoint_priority = i;
      i = MAX_COMMAND;
    }
  }
  if (setpoint_priority==-1) {
    fprintf( term, "couldn't find behavior setpoint on stack\n");
    return -1;
  }
  vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[0].x = arg[0];
  vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[1].x = arg[1];

  vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[2].x = arg[2];
  vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[3].x = arg[3];

  /* in case beh timed out */
  vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[6].x = 0;

  fprintf( term, "\nnew setpoint; %6g (rad) %6g (m) %g (m/sec) "
           "%6g (s)\n",
           vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[0].x,
           vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[1].x,
           vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[2].x,
           vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[3].x );
  fprintf( syslog, "\nnew setpoint; %6g (rad) %6g (m) %g (m/sec) "
           "%6g (s)\n",
           vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[0].x,
           vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[1].x,
           vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[2].x,
           vvp->sta[vvp->cstate].c[setpoint_priority].b.arg[3].x );
}
```

```
  return count;
}
```

A.3 SGI C Code

A.3.1 mcast.c

```
/*********************************************************************
 * mcast.c
 *
 * Multicast code for E-Z multicast programs.
 * John H. Kim, 28 August, 1995.
 *********************************************************************
 * HOW TO INCLUDE MULTICAST IN YOUR PROGRAM
 *
 * Step 1:  Include the header in your program (optional on SGI)
 * Step 2:  Use the function calls as described below
 * Step 3:  Compile mcast.c and link in mcast.o
 *********************************************************************
 * HOW TO CALL MULTICAST ROUTINES IN YOUR PROGRAM
 *
 * 1) Mcast_init(port); - initializes the multicast routine for your program.
 *    You can have more than one process receive on the same machine.
 *    Port can be 1 or 2.  Default values (change via .multicastrc file):
 *      port1 = 6000
 *      ip address1 = 224.0.0.250
 *      ttl1 = 0
 *      port2 = 6002
 *      ip address2 = 224.0.0.252
 *      ttl2 = 0
 *
 * 2) Mcast_set_nonblock(port); - sets up port for nonblocking read.
 *    If you don't use this, the Mcast_receive routine will wait for
 *    input before returning control to your program.
 *
 * 3) Mcast_send(char *msg, int msgsize); - multicasts a string.  String
 *    is contained in msg, it's length is msgsize (so you can transmit
 *    '\n' as well).  Use Mcast_send2 for port 2.
 * or
 * 3) Mcast_receive(char *msg, int msgsize, char *netaddress); - receives
 *    a multicast string into msg, max length msgsize. netaddress returns
 *    the IP address of the sending machine.  Use Mcast_receive2 for port 2.
 *
 *********************************************************************/

#include <stdio.h>
#include <string.h>
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include "mcast.h"

struct mcast_struct {
  unsigned short port;  /* Port number,   read in */
  char ip_group[16];    /* IP address,    read in */
  u_char ttl;           /* Time to live,  read in */

  struct sockaddr_in addr;  /* socket address                */
  int    addrlen, fd;       /* address length, socket handle */
  int on;                   /* multicast or not?             */
  struct ip_mreq mreq;      /* not sure */
} mcast, mcast2;

/* Reads in multicast configuration from .multicastrc file */
void Read_mcast_config(void)
{
  char buffer[256];                        /* readin buffer */
  char *value;                             /* pointer to middle of buffer */
  char *defaultname = "./.multicastrc";    /* rc file in current directory */
  char *homepath;                          /* rc file in home directory */
  char backupname[256];
  FILE *mcastfile;                         /* file pointer for above */

  homepath = getenv("HOME");
  strcpy(backupname, homepath);
  sprintf(backupname, "%s", homepath);
  strcat(backupname,"/.multicastrc");
  fprintf(stderr,"backupname = %s\n", backupname);

  /* Set defaults */
  mcast.port = 6000;                         /* set default port, */
  mcast2.port = 6002;                        /* set default port, */
  strcpy(mcast.ip_group, "224.0.0.250");     /* ip address, and   */
  strcpy(mcast2.ip_group, "224.0.0.252");    /* ip address, and   */
  mcast.ttl = 0;                             /* ttl.              */
  mcast2.ttl = 0;                            /* ttl.              */
  mcastfile = NULL;                          /* clear pointer */

  /* Find .multicastrc file and prep it for reading */
  if (mcastfile=fopen(defaultname, "r")) {
    fprintf(stderr,"Using .multicastrc file in current directory.\n");
  } else {
    if (mcastfile=fopen(backupname, "r")) {
      fprintf(stderr,"Using .multicastrc file in home directory.\n");
    } else {
      fprintf(stderr,"Couldn't find .multicastrc file.  Using defaults.\n");
    }
  }

  /* Read in settings found in .multicastrc file */
  if (mcastfile)
    while (fgets(buffer, 255, mcastfile) != NULL) { /* While there are lines */
      if (buffer[0] != '#') {                       /* Ignore if comment     */
        if (strncmp(buffer, "port=", 5) == 0) {     /* If it's a port number */
          value = &buffer[5];
          mcast.port = (unsigned short) atoi(value);/* set the port to value */
        } else if (strncmp(buffer, "address=", 8) == 0) {/* If it's an addr */
          value = &buffer[8];
          value[strlen(value)-1] = NULL;            /* get rid of \n         */
          strcpy(mcast.ip_group, value);            /* set address to value  */
        } else if (strncmp(buffer, "ttl=", 4) == 0) {/* If it's a ttl        */
          value = &buffer[4];
          mcast.ttl = (u_char) atoi(value);         /* set the ttl value     */
        } else if (strncmp(buffer, "port2=", 6) == 0) {/* If it's a port #   */
          value = &buffer[6];
          mcast2.port = (unsigned short) atoi(value);/* set the port to value*/
        } else if (strncmp(buffer, "address2=", 9) == 0) {/* If it's an addr */
          value = &buffer[9];
          value[strlen(value)-1] = NULL;            /* get rid of \n         */
          strcpy(mcast2.ip_group, value);           /* set address to value  */
        } else if (strncmp(buffer, "ttl2=", 5) == 0) {/* If it's a ttl       */
          value = &buffer[5];
          mcast2.ttl = (u_char) atoi(value);        /* set the ttl value     */
        } /* else */
      } /* if */
    } /* while */

  fprintf(stderr,"Port = %d, Address = %s, ttl = %d\n",
   mcast.port, mcast.ip_group, mcast.ttl);
  fprintf(stderr,"Port2 = %d, Address2 = %s, ttl2 = %d\n",
   mcast2.port, mcast2.ip_group, mcast2.ttl);
}

/* Initializes multicast networking stuff you don't want to worry about */
void Mcast_init(int portnum)
{
  Read_mcast_config(); /* Read in multicast configuration */
  /* printf("%d,%s,%d\n", mcast.port, mcast.ip_group, mcast.ttl); */


  if (portnum == 1) {
    mcast.on=1;
    mcast.fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (mcast.fd < 0) {
      perror("socket");
      exit(1);
    }
    bzero(&mcast.addr, sizeof(mcast.addr));
    mcast.addr.sin_family = AF_INET;
    mcast.addr.sin_addr.s_addr = htonl(INADDR_ANY);
    mcast.addr.sin_port = htons(mcast.port);
    mcast.addrlen = sizeof(mcast.addr);

    /* Receive */
    setsockopt(mcast.fd, SOL_SOCKET, SO_REUSEPORT, &mcast.on,
               sizeof(mcast.on));
    if (bind(mcast.fd, &mcast.addr, sizeof(mcast.addr)) < 0) {
      perror("bind");
      exit(1);
    }
    mcast.mreq.imr_multiaddr.s_addr = inet_addr(mcast.ip_group);
    mcast.mreq.imr_interface.s_addr = htonl(INADDR_ANY);
    if (setsockopt(mcast.fd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
                   &mcast.mreq, sizeof(mcast.mreq)) < 0) {
      perror("setsockopt mcast.mreq");
      exit(1);
    }
    setsockopt(mcast.fd, IPPROTO_IP, IP_MULTICAST_TTL, &mcast.ttl,
               sizeof(mcast.ttl));
    /* Send, must come after socket options */
    mcast.addr.sin_addr.s_addr = inet_addr(mcast.ip_group);
  } else if (portnum == 2) {
    /* port 2 */
    mcast2.on=1;
    mcast2.fd = socket(AF_INET, SOCK_DGRAM, 0);
    if (mcast2.fd < 0) {
      perror("socket");
      exit(1);
    }
    bzero(&mcast2.addr, sizeof(mcast2.addr));
    mcast2.addr.sin_family = AF_INET;
    mcast2.addr.sin_addr.s_addr = htonl(INADDR_ANY);
    mcast2.addr.sin_port = htons(mcast2.port);
    mcast2.addrlen = sizeof(mcast2.addr);

    /* Receive */
    setsockopt(mcast2.fd, SOL_SOCKET, SO_REUSEPORT, &mcast2.on,
               sizeof(mcast2.on));
    if (bind(mcast2.fd, &mcast2.addr, sizeof(mcast2.addr)) < 0) {
      perror("bind");
      exit(1);
    }
    mcast2.mreq.imr_multiaddr.s_addr = inet_addr(mcast2.ip_group);
    mcast2.mreq.imr_interface.s_addr = htonl(INADDR_ANY);
    if (setsockopt(mcast2.fd, IPPROTO_IP, IP_ADD_MEMBERSHIP,
                   &mcast2.mreq, sizeof(mcast2.mreq)) < 0) {
      perror("setsockopt mcast2.mreq");
      exit(1);
    }
    setsockopt(mcast2.fd, IPPROTO_IP, IP_MULTICAST_TTL, &mcast2.ttl,
               sizeof(mcast2.ttl));
    /* Send, must come after socket options */
    mcast2.addr.sin_addr.s_addr = inet_addr(mcast2.ip_group);
  } /* else */
}

void Mcast_set_nonblock(int portnum)
{
  /* nonblocking socket */
  if (portnum == 1) {
    if (fcntl(mcast.fd, F_SETFL, FNDELAY) < 0) {
      perror("fcntl F_SETFL, FNDELAY");
      exit(1);
    }
  } else if (portnum == 2) {
    if (fcntl(mcast2.fd, F_SETFL, FNDELAY) < 0) {
      perror("fcntl F_SETFL, FNDELAY");
      exit(1);
    }
  } /* else if */
}

int Mcast_send(char *msg, int msgsize)
{
  return (sendto( mcast.fd, msg, msgsize, 0, &mcast.addr, mcast.addrlen));
}

int Mcast_receive(char *msg, int msgsize, char *netaddress)
{
  int result;
  result = recvfrom(mcast.fd, msg, msgsize, 0, &mcast.addr, &mcast.addrlen);
  strcpy(netaddress, inet_ntoa(mcast.addr.sin_addr));
  return (result);
}

int Mcast_send2(char *msg, int msgsize)
{
  return (sendto( mcast2.fd, msg, msgsize, 0, &mcast2.addr, mcast2.addrlen));
}

int Mcast_receive2(char *msg, int msgsize, char *netaddress)
{
  int result;
  result = recvfrom(mcast2.fd, msg, msgsize, 0, &mcast2.addr, &mcast2.addrlen);
  strcpy(netaddress, inet_ntoa(mcast2.addr.sin_addr));
  return (result);
}
```

A.3.2 network.c

This routine is essentially identical to `comm.c` except it works with the vehicle simulator. In order to maintain future cross-platform compatibility, it compiles into the main simulator, rather than linking to the data structure like `comm.c`. Instead of relaying data and commands between the modem and data structure, it listens and transmits directly to the network, using the previous multicasting routines. The `report` routine demonstrates the differences with `comm.c`

```
  ⋮

int report(int *out_ind, int num_out )
{
  int i, cnt, msgsz=0;
  int uniform_format = 1;

  static int print_names = 1;
  char msg[500] = "", buf[50];
  unsigned char chksum=0;

  /* the check sum is everything up to but not including the colon */
  if (print_names) {
    print_names=0;
    sprintf(msg, "NAMES> ");
    for (i=0;i<num_out;i++) {
      strcat(msg, vvp->s[out_ind[i]].name);
      strcat(msg, space);
    }
    /* attach checksum and then cr_lf */
    chksum = calc_check_sum( msg );
    strcat(msg, colon);
    strcat(msg, space);
    sprintf( buf, "%02X", chksum);
    strcat(msg, buf);
    strcat(msg, crlf);
    strcat(msg, space);
    msgsz = strlen( msg );
    cnt = Mcast_send( msg, msgsz );
    if (cnt < 0) {
      fprintf( stderr, "Error sending message, Mcast_send "
               "returns %d, errno %d\n", cnt, errno);
    } /* if cnt < 0 */
  }

  sprintf(msg, "VALUES> ");
  for (i=0;i<num_out;i++) {
    if (uniform_format)
      sprintf( buf, "%6.3lf ", vvp->s[out_ind[i]].x);
    else
      sprintf( buf, "%.5g ", vvp->s[out_ind[i]].x);
    strcat( msg, buf);
  }
  chksum = calc_check_sum( msg );
  strcat(msg, colon);
  strcat(msg, space);
  sprintf( buf, "%02X", chksum);
  strcat(msg, buf);
  strcat(msg, crlf);

  fprintf( stderr, "network_report: %s", msg );
  msgsz = strlen( msg );
  cnt = Mcast_send( msg, msgsz );
  if (cnt < 0) {
    fprintf( stderr, "Error sending message, Mcast_send "
             "returns %d, errno %d\n", cnt, errno);
  } /* if cnt < 0 */
}

⋮
```

Bibliography

[1] K. Asakawa, J. Kojima, Y. Ito, S. Takagi, Y. Shirasaki, and N. Kato. Autonomous underwater vehicle Aqua Explorer 1000 for inspection of underwater cables. In *IEEE Oceans*, pages 10–17, 1996.

[2] D. K. Atwood, J. J. Leonard, J. G. Bellingham, and B. A. Moran. An acoustic navigation system for multiple vehicles. In *Proc. Int. Symp. on Unmanned Untethered Submersible Technology*, pages 202–208, September 1995.

[3] G. Ayela and J. M. Coudeville. TIVA: A long range, high baud rate image/data acoustic transmission system for underwater applications. In *Underwater Defence Technolgy Conf.*, 1991.

[4] J. G. Bellingham and T. R. Consi. State configured layered control. In *Proceedings of the IARP 1st Workshop on Mobile Robots for Subsea Environments*, pages 75–80, Monterey, CA, USA, October 1990.

[5] J. G. Bellingham, T. R. Consi, R. Beaton, and W. Hall. Keeping layered control simple. In *Proceedings AUV '90*, 1990.

[6] J. G. Bellingham, T. R. Consi, U. Tedrow, and D. Di Massa. Hyperbolic acoustic navigation for underwater vehicles: Implementation and demonstration. In *Proceedings AUV '92*, pages 304–309, 1992.

[7] J. G. Bellingham, C. A. Goudey, T. R. Consi, J. W. Bales, D. K. Atwood, J. J. Leonard, and C. Chryssostomidis. A second generation survey AUV. In *IEEE Conference on Autonomous Underwater Vehicles*, Cambridge, MA, 1994.

[8] J. G. Bellingham, C. A. Goudey, T. R. Consi, and C. Chryssostomidis. A small, long-range autonomous underwater vehicle for deep ocean exploration. In *Proc. International Society of Off-shore and Polar Engineers*, 1992.

[9] J. G. Bellingham and J. J. Leonard. Task configuration with layered control. In *Proceedings of the IARP 2nd Workshop on Mobile Robots for Subsea Environments*, pages 193–302, Monterey, CA, USA, May 1994.

[10] D. Bertsekas and R. Gallager. *Data Networks.* Prentice Hall, 1992.

[11] D. Brady and J. Catipovic. Adaptive multiuser detection for underwater acoustic channels. *IEEE J. Ocean Engineering*, 19(2):158–165, April 1994.

[12] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[13] R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, pages 3–15, June 1990.

[14] D. P. Brutzman. *A Virtual World for an Autonomous Underwater Vehicle.* PhD thesis, Naval Postgraduate School, Monterey, CA, December 1994.

[15] D. P. Brutzman. Internet Protocol over seawater (IP/SW): Towards interoperable underwater networks. In *Proc. Int. Symp. on Unmanned Untethered Submersible Technology*, pages 444–457, 1995.

[16] C. Fang C. Eck, D. Porta. High speed digital acoustic telemetry system. In *Proc. Int. Symp. on Unmanned Untethered Submersible Technology*, pages 348-362, 1987.

[17] J. Catipovic. Performance limitations in underwater acoustic telemetry. *IEEE J. Ocean Engineering*, 15(3):205–216, July 1990.

[18] J. Catipovic, D. Brady, and S. Etchemendy. Development of underwater acoustic modems and networks. *Oceanography*, 6(3):112, April 1994.

[19] S. G. Chappell, J. C. Jalbert, P. Pietryka, and J. Duchesny. Acoustic communications between two autonomous underwater vehicles. In *IEEE Oceans*, pages 462–469, 1994.

[20] DIS Steering Committee. The DIS Vision- a map to the future of distributed simulation, 1994. http://ftp.sc.ist.ucf.edu/STDS/docs/.

[21] T. B. Curtin, J. G. Bellingham, J. Catipovic, and D. Webb. Autonomous ocean sampling networks. *Oceanography*, 6(3):86–94, 1993.

[22] Datasonics Inc., Cataumet, MA. Datasonics ATM-800 series modem user's manual.

[23] I. Dyer. Fundamentals and applications of underwater sound, 1995. Course notes for MIT Dept. of Ocean Engineering class 13.851.

[24] J. Catipovic (editor). Special issue on acoustic communications. *IEEE J. Ocean Engineering*, 21(2):125–192, April 1996.

[25] M. Evans. Phobos mars probe, http://leonardo.jpl.nasa.gov/msl/QuickLooks/phobosQL.html.

[26] G. C. Feldman and T. C. Viola. The Jason Project: The NR-1 Submarine Revealed. http://www.popsci.com/content/science/features/jason/jason.2.html.

[27] M. Johnson, D. Herold, and J. Catipovic. The design and performance of a compact underwater acoustic network node. In *IEEE Oceans*, volume 3, pages 467–471, September 1994.

[28] V. A. Kaharl. *Water Baby - The Story of Alvin*. Oxford University Press, 1990.

[29] A. Kaya and S. Yauchi. An acoustic communication system for subsea robot. In *IEEE Oceans*, pages 765–770, Oct. 1989.

[30] J. H. Kim, B. A. Moran, J. J. Leonard, and J. G. Bellingham. Experiments in remote monitoring and control of autonomous underwater vehicles. In *IEEE Oceans*, page ????, 1996.

[31] J. S. Merriam L. E. Freitag. Robust 5000 bit per second underwater communication system for remote applications. In *MTS'90 Conf.*, Feb. 1990.

[32] R. L. Eastwood L. E. Freitag, J. A. Catipovic. Acoustic communications system for the AMMT program. In *IEEE Oceans*, volume Supplement, pages 87–92, 1996.

[33] J. J. Leonard, S. T. Tuohy, J. G. Bellingham, B. A. Moran, J. H. Kim, H. Schmidt, N. M. Patrikalakis, and C. Chryssostomidis. Virtual environments for AUV development and ocean exploration. In *Proc. Int. Symp. on Unmanned Untethered Submersible Technology*, pages 436–443, September 1995.

[34] J. Locke. An introduction to the internet networking environment and SIMNET/DIS. http://www-npsnet.cs.nps.navy.mil/npsnet/publications.html, 1993.

[35] J. G. Proakis M. Stojanovic, J. A. Catipovic. Phase-coherent digital communications for underwater acoustic channels. *IEEE J. Ocean Engineering*, 19:100–112, 1994.

[36] T. Sasaki M. Suzuki. Digital acoustic image transport system for deep sea research submersible. In *IEEE Oceans*, volume III, pages 567–570, Oct 1992.

[37] M. R. Macedonia and D. P. Brutzman. MBone provides audio and video across the internet. *IEEE Computer*, 27(4):30–36, 1994.

[38] G. R. Mackelburg. Acoustic data links for UUVs. In *IEEE Oceans*, pages 1400–1406, 1991.

[39] P. Maes. Modeling autonomous agents, 1996. Course notes for MIT Dept. of Media Arts and Sciences class MAS738.

[40] J. B. Newman. Fiber-optic data network for the ARGO/JASON vehicle system. *IEEE J. Ocean Engineering*, 15(2), April 1990.

[41] G. Pappas, W. Shotts, M. O'Brien, and W. Wyman. The DARPA/Navy unmanned undersea vehicle program. *Unmanned Systems*, 9(2):24–30, 1991.

[42] Proxim Inc., Mountain View, CA. *ProximLink PL and PL2 Radio Modem Family User's Manual*, June 1994. (415)-960-1630.

[43] J. A. Catipovic R. L. Eastwood, L. E. Freitag. Compression techniques for improving underwater acoustic transmission of images and data. In *IEEE Oceans*, volume Supplement, pages 63–68, 1996.

[44] H. Schmidt and J. G. Bellingham. Real-time frontal mapping with AUVs in a coastal environment. In *IEEE Oceans*, pages 1094–1098, 1996.

[45] Richard W. Stevens. *Unix Network Programming*. Prentice Hall, 1990.

[46] M. Stojanovic. Recent advances in high-speed underwater acoustic communications. *IEEE J. Ocean Engineering*, 21:125–136, 1996.

[47] J. Travis. Deep-sea debate pits Alvin against Jason. *Science*, 259:1534–1536, 1993.

[48] R. Uhrich and J. M. Walton. AUSS - Navy Advanced Unmanned Search System. *Sea Technology*, pages 29–35, February 1993.

[49] R. Uhrich and J. M. Walton. Supervisory control of unthethered undersea systems: A new paradigm verified. In *Proc. Int. Symp. on Unmanned Untethered Submersible Technology*, pages 1–5, 1995.

[50] J. M. Walton. Advanced unmanned search system. In *IEEE Oceans*, pages 1392–1399, 1991.

[51] S. J. Watson and R. W. Uhrich. Advanced unmanned search system image compression and data format. In *IEEE Oceans*, pages 1407–1413, 1991.

[52] G. M. Wenz. Acoustic ambient noise in the ocean: Spectra and sources. *J. Acoustic Sociecty of America*, 34:1936–1956, 1962.

[53] D. R. Yoerger, A. M. Bradley, and B. B. Walden. The Autonomous Benthic Explorer (ABE): An AUV optimized for deep seafloor studies. In *Proc. Int. Symp. on Unmanned Untethered Submersible Technology*, pages 60–69, 1991.