NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

<div style="border:1px solid">

ANALYSIS AND IMPROVEMENT OF AN ULTRASONIC SONAR SYSTEM ON AN AUTONOMOUS MOBILE ROBOT

by

Jane Thayer Lochner

December 1994

Thesis Advisor: Yutaka Kanayama
Thesis Co-Advisor: Donald Walters

</div>

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE ANALYSIS AND IMPROVEMENT OF AN ULTRASONIC SONAR SYSTEM ON AN AUTONOMOUS MOBILE ROBOT	5. FUNDING NUMBERS

6. AUTHOR(S) Jane Thayer Lochner

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

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

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*
This research addresses the problems experienced by the autonomous mobile robot, Yamabico-11, with its ultrasonic sonar system. It explains the basics of acoustic theory as related to Yamabico-11 and explains the sources of limitations imposed on Yamabico-11 by the physical nature of the problem. This paper documents the basic characteristics of the sonar hardware and examines causes of sonar range errors. Finally, this research leads to improvements of the current sonar system to provide better directional coverage through a new sonar configuration.

14. SUBJECT TERMS Robotics, Sonar, Sensors, Acoustics	15. NUMBER OF PAGES 120
	16. PRICE CODE

17. SECURITY CLASSIFI-CATION OF REPORT Unclassified	18. SECURITY CLASSIFI-CATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFI-CATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

ANALYSIS AND IMPROVEMENT OF AN ULTRASONIC SONAR SYSTEM ON AN AUTONOMOUS MOBILE ROBOT

by

Jane T. Lochner
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1984
M.S., National University, 1990

Submitted in partial fulfillment of the requirements for the degrees of

MASTER OF SCIENCE IN APPLIED PHYSICS
MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1994

iv

ABSTRACT

This research addresses the problems experienced by the autonomous mobile robot, Yamabico-11, with its ultrasonic sonar system. It explains the basics of acoustic theory as related to Yamabico-11 and explains the sources of limitations imposed on Yamabico-11 by the physical nature of the problem. This paper documents the basic characteristics of the sonar hardware and examines causes of sonar range errors. Finally, this research leads to improvements of the current sonar system to provide better directional coverage through a new sonar configuration.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

A. SPATIAL REASONING PHILOSOPHY

Good spatial reasoning is critical to the success of any mobile robotics project; it allows a robot to interpret what it "sees" and to perform intelligent motions. Spatial reasoning requires the robot to create and maintain a cognitive map, or knowledge structure, of its world. There are two main schemes employed by robotics projects to represent this cognitive map: occupancy arrays and constructive solid geometry.

The implementation of an occupancy array representation in 2-D is similar to the implementation of a graphic picture. Grids, similar to the pixels of a computer monitor, divide the world. The labels, occupied, empty or unknown, similar to the red, green, blue (RGB) designations used in graphics, give the status of each grid Additionally, each grid has an uncertainty factor between 0 and 1, similar to the 0 - 255 value assigned to the RGB colors. This uncertainty factor arises due to incomplete sensor data and/or partially occupied grids. Humans can visualize occupancy arrays easily; the occupancy arrays easily transform into a graphic picture. However, they do have some drawbacks. If an object moves, the uncertainty of its position increases dramatically. Furthermore, occupancy arrays require a significant amount of space, depending on the granularity of the representation. For example, suppose the robot requires 2-D knowledge of its world to within one centimeter and its world is a five meter square box. This world is represented by a 1000 x 1000 array containing one million grids; Each element of the array would need to maintain information on both the status and uncertainty of the grid.(Davis, 1990, pp. 264-270) Many robotics projects, such as the Neptune, Terregator and Uranus robots at Carnegie-Mellon University, use occupancy arrays to describe their robot's world (Elfes, 1987, p. 255).

The other common implementation of the cognitive map uses Constructive Solid Geometry (CSG). CSG represents complex objects as a combination of fundamental shapes. The fundamental shapes used depends on the domain of the world. Each shape has its own, or local, coordinate system

1

and this coordinate system has a position relative to the world, or global, coordinate system. The description of an object consists of the appropriate dimensions and the location and orientation of the local coordinate system for each fundamental shape forming the object. This works fine for describing a known world, but the method breaks down when an unknown obstacle is encountered. The robot is unable to determine neither the dimensions nor the location of the local coordinate system of each fundamental shape needed to compose the object; the robot only knows information about the visible boundary of the object and can not determine to what shape this boundary belongs.(Davis, 1990, pp. 270-273)

B. SENSORS

Intelligent management of on board sensors is critical to successful robot navigation. Some of the basic types of sensors used in robotics include sonars, Charge-Coupled Device (CCD) cameras, laser range finders and tactile sensors. Each sensor type has its own set of strengths and limitations which needs to be considered when designing a sensor suite. The sensor suite should use complementary sensors and should only use those sensors which are appropriate for the particular application. Correlating data from multiple sensors is an extremely difficult problem that many organizations and universities, including this one, are trying to solve. The problem is complicated even further when different types of sensors provide the data. However, the benefits of this sensor fusion are tremendous.

Ultrasonic sonars probably are one of the most widely used sensors due to their low cost and ease of implementation. They provide good range resolution, but the bearing information is limited. Ultrasonic sonars are excellent at detecting the presence of an object, but cannot determine object size without the maneuvering of their platform.

On the other hand, CCD cameras can provide boundary information about an object. However, image processing is expensive in both dollar cost, Central Processing Unit (CPU) time, disk storage and power requirements. The lack of depth perception is a major disadvantage which can be overcome by using multiple CCD cameras, reference images, or other similar strategies.

2

Laser range finders use the same basic principles as ultrasonic sonars; they measure the time of flight to determine distance. However, since laser range finders use the speed of light (3×10^8 m/s), their response time is much better than that of ultrasonic sonars which use the speed of sound (343 m/s). They are not used on smaller robotics projects because they have a higher price tag and require more power.

Tactile sensors are used mainly with manipulators, but have found uses with mobile robots. They have been used on the feet of walking robots to facilitate foot placement and on the periphery of wheeled robots, similar to curb feelers installed on some cars.

C. YAMABICO-11 PHILOSOPHY

Yamabico-11 is a research robot at the Naval Postgraduate School whose purpose is to implement and validate new theories in robotics, including motion control and spatial reasoning. Consequently, its configuration, both hardware and software, is continually evolving.

The Yamabico-11 project uses a method similar to CSG to describe its world; the world and the objects in it are depicted by polygons. Known objects are fitted to polygons located within the inverted polygon representing Yamabico-11's world. As Yamabico-11 moves within its world, it uses a least-squares fitting algorithm to fit its positional sonar readings to line segments. Yamabico-11 matches these line segments to its known world; if no known object corresponds to the sonar data, the line segments are stored as a new object. This method has advantages over occupancy arrays and CSG because it requires less storage space and does not need to know information about the location and orientation of the local coordinate system of objects. Fitting sonar data to line segments reduces the impact of partial and/or erroneous data. Objects, both known and unknown, are represented by the global position of each vertex of the polygon or endpoints of the line segment.

Yamabico-11 uses and array of twelve ultrasonic sonars as its main sensor system. Since Yamabico-11 operates in a controlled environment, it does not need a sophisticated, long-range sensor system. Since Yamabico-11 also serves as a teaching tool, it includes a CCD camera to provide a means for

image processing research. Since image processing is a CPU intensive operation, the CCD camera has not been incorporated as an integral part of the sensor suite. Future plans for Yamabico-11 development include the integration of the CCD camera.

The quality of the data received depends on the sensor system characteristics. This research work examines the quality of the data produced by sonars and shows how to use this data for intelligent obstacle avoidance.

II. PROBLEM STATEMENT AND APPROACH

A. PURPOSE

The purpose of this thesis work was to:

1. *Determine the acoustic characteristics of the current ultrasonic sensors and the effects of the current sensor configuration on the capabilities and limitations of Yamabico-11.* It will explain the origin of the observations made so far and will help to develop corrections to the sensor data processing to adjust for the physical phenomena.

2. *Improve the capability of Yamabico-11 to navigate autonomously around unknown obstacles.*

B. APPROACH

1. Acoustic Characteristics

The approach followed during this research work was to:

a. Determine the theory of the physical phenomena related to the ultrasonic sensors.
b. Test and verify the theoretical predictions in the laboratory setting.
c. Make recommendations for improvements.
d. Implement improvements.
e. Test the improvements using Yamabico-11 as the test bed.

2. Robot Navigation

Intelligent robot navigation requires interfacing the sensor system(s) and the path planning module to derive a workable algorithm for obstacle avoidance. This task was accomplished by:

a. Developing a dynamic function to determine a safe path for avoiding an obstacle, assuming either the vertices of the object are known or the width can be determined.

b. Testing the ability of Yamabico-11 to avoid obstacles autonomously using this function in a variety of scenarios.

III. YAMABICO-11

A. HISTORY

Yamabico-11 is an autonomous mobile robot powered by two 12-volt batteries and driven on two wheels by DC motors. These motors drive and steer the wheels while four shock absorbing caster wheels balance the robot. It uses twelve 40 kHz ultrasonic sensors to sense its environment. Recently, its master processor was upgraded from the Motorola MC68020 microprocessor to the SPARC4 microprocessor. This upgrade to the SPARC4 microprocessor expanded Yamabico-11's memory and changed the development environment. The high-level Model-based Mobile-robot Language (MML) software, written in ANSI C, is compiled using a "Makefile" and then downloaded to Yamabico-11. An onboard laptop console (Macintosh 145 Powerbook) provides real-time command level communication between the user and the robot.

The Motorola microprocessor on Yamabico-11 uses MML Version 10 (MML 10) and requires compilation using the "gravy6" server in the Computer Science Department at the Naval Postgraduate School. MML 10 consists of a kernel and a user program. The kernel contains the compiled code for the robot's application software. The user program uses the MML functions to control the robot. Once compiled, these programs are downloaded to the robot via an RS-232 link at a baud rate of 19,200. Typing the command "lo=dluk" at the prompt on the laptop downloads both the kernel and the user program to Yamabico-11; typing "lo=dlu" downloads just the user program. Once downloaded, the user types "g 304000" to run the user program.

The SPARC4 microprocessor on Yamabico-11 uses MML Version 11 (MML 11). Compilation of MML 11 uses the GNU 'C' Compiler available on the SPARC workstations and downloading uses an Ethernet cable connected to the "libra" server in the Computer Science Department at the Naval Postgraduate School. The "Makefile" in MML 11 compiles the kernel files and the user files into a single executable file called "user" and copies this file into the "SPARC4/target" directory in the "yamabico" account. The "libra" server automatically checks this directory every minute to see if the user program has changed and updates its files accordingly. This step is required to ensure the

security and integrity of the "libra" server. To load the executable program, the user types "bootp" at the prompt on the laptop. If the user enters "bootp" at the laptop prompt before the "libra" server has updated its files, the new compiled program will not be available yet, so the old compiled program will be loaded. Therefore, it is important that the user wait at least one minute after compiling the program before downloading the program to Yamabico-11. Once loaded, the user types "run" to execute the user program.

Navy Lieutenant Scott Book developed MML 11 in March 1994. MML 10 became cumbersome with the addition of new functions. MML 10 relied heavily upon numerous global variables and did not adhere to standard software engineering practices. Although MML 11 allows Yamabico-11 to use the SPARC4 microprocessor, the main thrust of Lieutenant Book's work was the restructuring of the MML to eliminate global variables and secondly to develop guidelines for future programming. Lieutenant Book successfully implemented and tested the motion control functions in MML 11. In September 1994, Navy Lieutenant Commander Frank Kelbe converted the sonar functions from MML 10 to MML 11.

B. SONAR SYSTEM

1. Hardware System

Yamabico-11's sonar system has been evolving since 1980. The original design consisted of an array of twelve ultrasonic transmitter/receiver pairs, hereafter referred to as sonar pairs, mounted around the periphery of the robot as per Figure 1, approximately a foot from the floor. The self-contained sonar system ran on a VME motherboard and interfaced with the Yamabico-11's Central Processing Unit (CPU) via the VME bus.

a. Sensor Configuration

In the original design, the twelve ultrasonic sonar pairs were divided into three logic control groups having each of their sensors located at 90 degree angles from each other: group 0 consists of pairs 0, 2, 5 and 7; group 1 consists of pairs 1, 3, 4 and 6; and group 2 consists of pairs 8, 9, 10 and 11. This grouping allowed four ultrasonic sonar pairs to operate simultaneously without interference. (Sherfey, 1991, pp. 10-11) Additionally, the sonar pairs
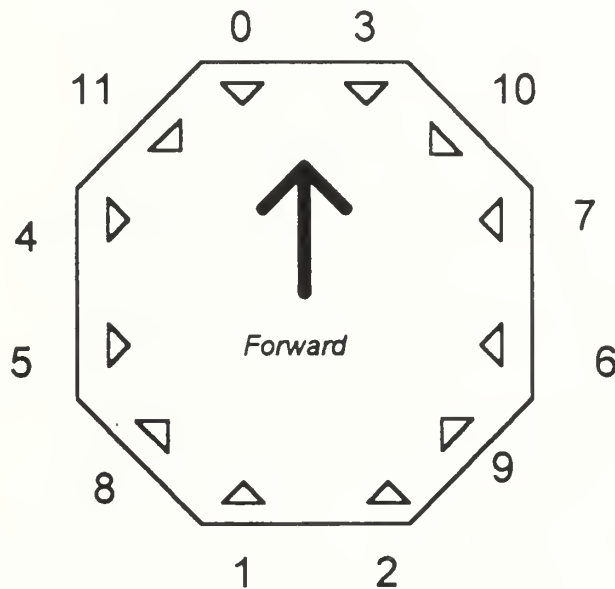
Figure 1
Ultrasonic Sensor Pair Location

were physically grouped in order to distribute the electrical load over the driver boards evenly. Sonar pairs 0, 2, 8 and 11 were on sonar driver board 1; sonar pairs 4, 6, 7 and 5 run off of sonar driver board 2 while sonar pairs 1, 3, 9 and 10 work from sonar driver board 3. Figure 2 shows the relationship between the different sonar pairs, the sonar motherboard and Yamabico-11's CPU.
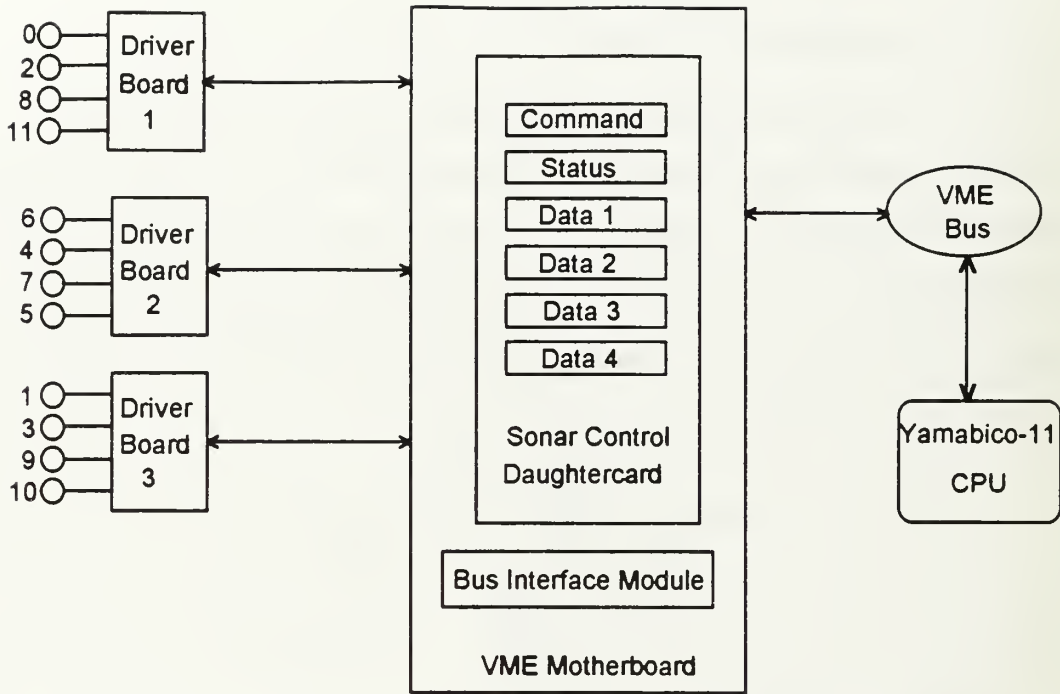
Figure 2
Sonar Hardware Architecture

b. Clock Counter

A clock counter data register computed the distance to a target.
The first 12 bits of the data register were reserved for the range data. The data
register kept track of the number of clock cycles expired between the transmit
and receive pulses. A clock cycle occurred every 6 microseconds. For TP2*,
the clock counter was enabled one cycle, or 25 microseconds, after the transmit
pulse begins; for TP1*, the clock counter was enabled one-and-a-half cycles, or
37.5 microseconds, after the transmit pulse begins. When a return signal
exceeded a set threshold at the receiver, the last value of the clock was copied
into the appropriate data register and the clock counter continued until ranging
was completed for all sonars in the group.

c. Sonar Range Calculation

The minimum range was based on the receiver being disabled during the transmit pulse. This prevented the receiver from being triggered by crosstalk from the transmitted signal. The pulse was transmitted for 0.5 milliseconds; at 343 meters per second, sound traveled a total of 17.15 centimeters in this time, but since this represented the two-way travel distance, the minimum theoretical range was one-half of this number or 8.575 centimeters. However, additional time was needed to accommodate circuitry switching and settling; therefore, in practice, firmware set the minimum range at 9.6 centimeters (Sherfey, 1991, p. 12).

The maximum range was a function of the hardware design. The maximum number that could be represented by 12 bits was $2^{12} - 1 = 4095$. At 6 microseconds per clock tick, this equated to $(6.0x10^{-6}) \times (4095) = 0.02457$ seconds. In 24.57 milliseconds, a sound wave could travel 8.428 meters. Since the sound wave must travel both out and back, the maximum one-way distance was one-half of this distance, or 4.214 meters. Therefore, due to system configuration constraints, Yamabico-11's sensors had an operating range of 9.6 centimeters to 4.214 meters. (Sherfey, 1991, pp. 11-13)

d. Sonar Driver Board

Within each logical group, two sensors were driven by one driver board while the other two sensors were driven by another driver board. The driver board produced a 0.5 millisecond transmit signal consisting of 20 cycles of a 40 kHz, 4.5 V peak-to-peak square wave. The driver board produced two signals, TP1* and TP2*. TP2* lagged TP1* by 180 degrees, allowing the driver board to power only one of the sensors at a time. The sonar control board interrupted Yamabico-11's central processing unit only when data was available from the sonar array.

The received signal was considerably weaker than the transmitted pulse so it was sent through a two-stage amplification circuit and then to a 74LS14 Schmitt Trigger which, given a variable input voltage, produced a constant output voltage signal. However, the Schmitt Trigger required a

minimum of 1.4 Volts to operate (Michiue, 1994, p. 32). The first stage of the amplification circuit yielded a voltage gain of 40 dB and the second stage a 12.87 dB gain for a total gain of 52.87 dB or 440 (Michiue, 1994, p. 23). Therefore, a minimum 6 mV peak-to-peak signal at the receiver was required to recognize the return signal.

The actual transmitted pulse was a packet of 20 individual pulses of equal amplitude, separated by 25 microseconds. Because of the finite bandwidth of the receiver transducer and preamplifier, the output amplitude of the receiver circuitry increased linearly to reach a maximum according to the equation

$$V = \frac{V_{max}}{5 \times 10^{-4}} * t \qquad \text{(Eq. 3-1)}$$

where t is the time, V is amplitude of the received voltage and V_{max} is the maximum voltage reached. The output reached a maximum at t = 0.5 milliseconds and then fell off according to the equation

$$V = V_{max}e^{-t/\tau} \qquad \text{(Eq. 3-2)}$$

where V is the voltage at the receiver, V_{max} is the maximum voltage reached, τ = 0.5 milliseconds and t is the time. Figure 3 gives a representation of the two pulse packets. The Schmitt Trigger fired once the amplitude of the received signal after amplification, V, reached 1.4 Volts. The time that this took varied because the maximum received signal strength, V_{max}, was a function of both the distance to and the reflectance of the object ensonified.

e. Accuracy

The clock counter - distance conversion algorithm should account for the hardware side effects. At room temperature (20°C) the speed of sound in air is 343 m/s. Using this value, the delay in starting the clock meant that the true range was about 0.43 or 0.64 centimeters longer, depending on which signal, TP1* or TP2*, was used to drive the transmitter. Since the last clock counter value was copied to the data register upon the receipt of a return signal, the value could be as much as one clock cycle, or 6 microseconds, off, causing

Pulse Packets

Transmit   Receive
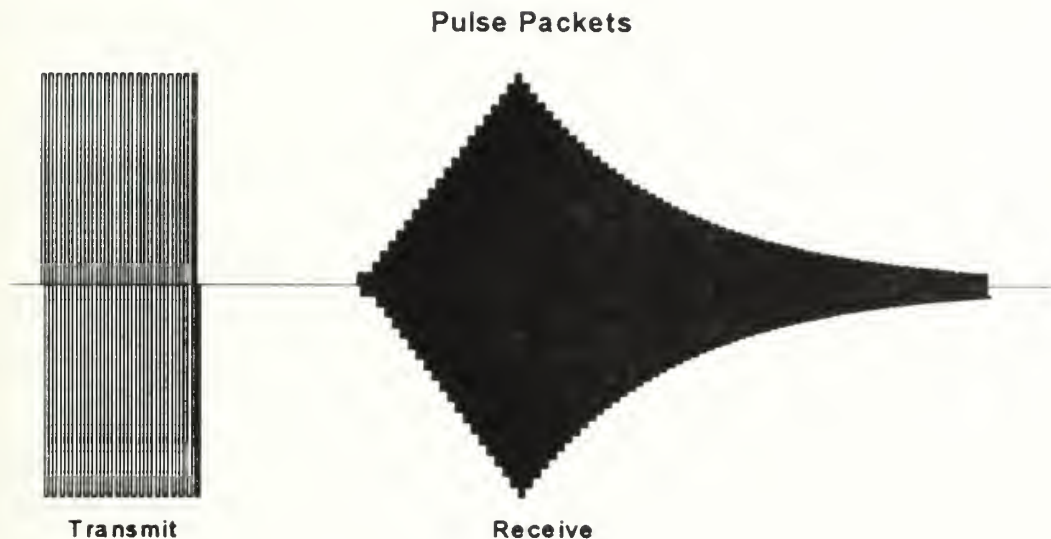
Figure 3
Representative Pulse Packets

the true range to be longer yet by up to another 0.1029 centimeters. Normally, one assumed that the actual object was ensonified by the leading edge of the transmit pulse. However, when using a detection threshold and the Schmitt Trigger, detection could occur anywhere within the received pulse. If the maximum amplitude of the received pulse was not detected, it was unlikely that any of the remaining received pulses would be detected. The detection could be delayed for up to 0.5 milliseconds, the transmit pulse width. This detection delay equated to an addition of up to 8.575 centimeters to the true range. Therefore, the worst-case accuracy of the sonar was the sum of the delays or about 9.1 centimeters.

f. Upgrades

Although the maximum theoretical range was set at 4.214 meters by the register size, in practice, the maximum achievable range was much less due to the sensitivity of the old sensors and circuitry. The front sensors were

upgraded in 1993 in hopes of overcoming the detection problems at long ranges. The new sensors are the Nicera T40-16 transmitter and the Nicera R40-16 receiver. The physical diameter of these sensors was 16.2 millimeters and they had an internal aperture diameter of 7.0 millimeters.

However, these sensors still were being driven by a 5 volt peak-to-peak supply voltage which produced an output voltage of about 4 volts peak-to-peak. At long ranges, the received signal often fell below the 6 mV required to fire the Schmitt Trigger, making detection nearly impossible. Consequently, in June 1994, the supply voltage to the transmitters was increased from 5 to 12 volts peak-to-peak to improve the capability of the new sensor system to detect obstacles consistently at long ranges. However, at this increased supply voltage, spillover became a problem. The amplitude of the spillover detected by the receiver was halved within the first millisecond after transmission and settled out after 6 milliseconds. To minimize the transmitter spillover effects, the receiver circuitry was redesigned to decrease the sensitivity during the first millisecond after transmission. Figure 4 shows this new sonar driver board configuration. (Michiue, 1994, pp. 10-13)

2. Software System

Yamabico-11's sensors can provide information about the surrounding environment about which it is unaware or can verify conditions that are already known. Obstacle detection and localization generally refers to gathering information about unknown objects. Sonar returns from obstacles are unplanned events. Alternatively, the sonar can be pre-programmed to acquire sonar returns based on the robot's knowledge of the world provided by its cognitive map and the robot's desired path. Ideally, it would continually look everywhere to determine its location within the world, but power requirements and signal interference patterns prevent this. (Yamabico manual, pp. 22-23).

The sonar system is used to navigate within Yamabico-11's known world. The sensors can return either the raw range data measured from the sonar pair face, or return the x - y coordinates of the sonar return in the robot's global coordinate system. Figure 5 shows the relation between the robot's local
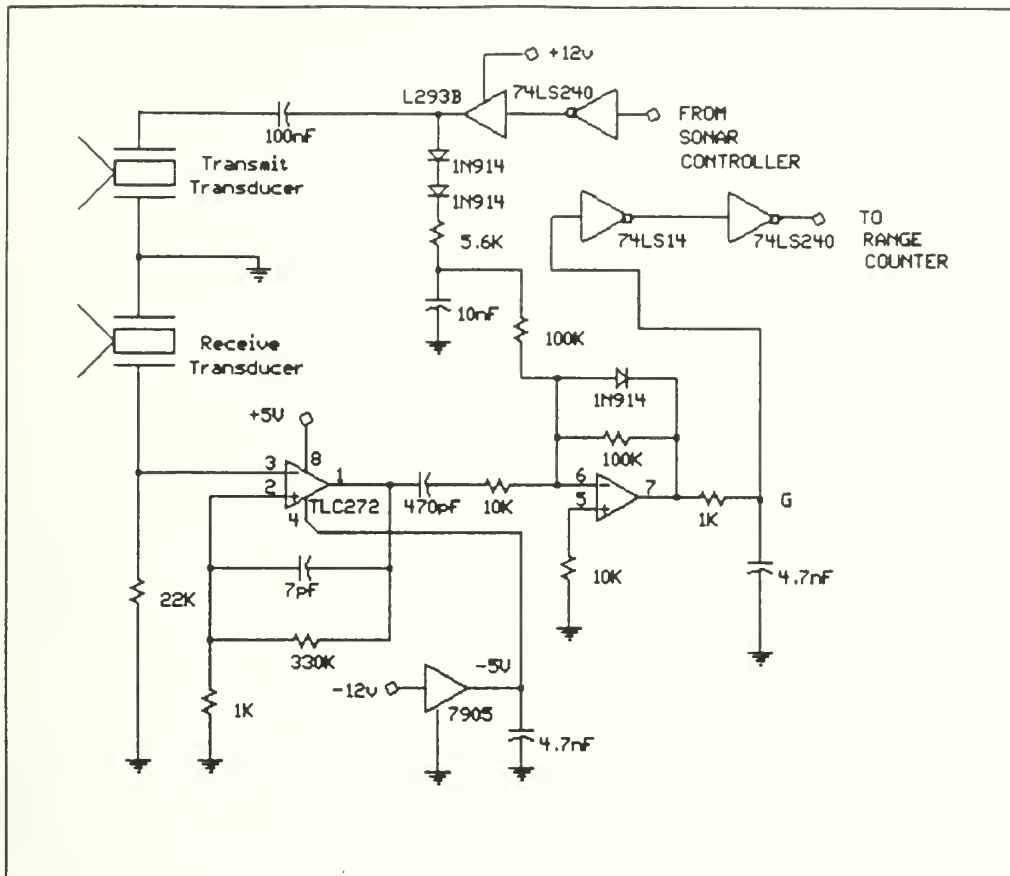
Figure 4
Sonar Driver Board Configuration

coordinate system, represented by the *x* and *y* axes, and its global coordinate system denoted by the **X** and **Y** axes.

Polygonal vertices in the global coordinate system describe the boundaries of the world and known objects. Sonar returns are depicted in Yamabico-11's local coordinate system, then transformed into the global coordinates. Once transformed, a linear fit to the sonar return allows a comparison with the objects in the known world to determine and/or verify the location of Yamabico-11. Additionally, the sonar system detects objects in the robot's path, but localization of these objects has not yet been accomplished.

The user had to tell the robot how and when to use its sonar through the user program developed in the MML. The user could enable/disable each of the twelve sonar pairs individually and could indicate the desired type of sonar

15
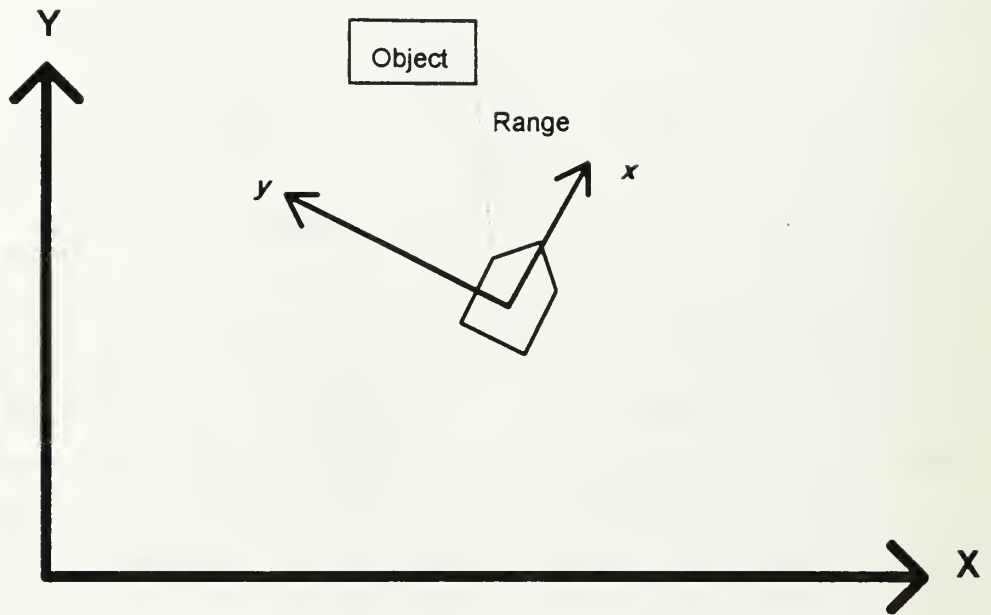
Figure 5
Global and Local Coordinate Systems Relationship

return data (raw range information or global coordinates) for each sonar pair. Additionally, the user could instruct Yamabico-11 to perform linear fitting of the sonar data. In 1993, Navy Lieutenant Patrick Byrne restructured the sonar system software kernel files to make them more modularized and user-friendly.

In MML10, the mnemonics were set up to make it easier for the user to operate the sonars. Table 1 lists these mnemonics. For example, a user program using these mnemonics may have contained the following ANSI C code fragment using MML10:

```
enable(FRONTR);          /*Turn on sonar 3*/
enable (FRONTL);         /*Turn on sonar 0*/
distance = sonar(FRONTR); /*Get raw range data from sonar 3*/
point = global(FRONTL);  /*Get global coordinates of data from
                           sonar 0*/
disable(FRONTR);         /*Turn off sonar 3*/
```

16

Mnemonic	Sonar	Group
FRONTL	0	0
FRONTR	3	1
LEFTF	4	1
LEFTB	5	0
RIGHTF	7	0
RIGHTB	6	1
BACKL	1	1
BACKR	2	0
BACKLEFT	8	2
BACKRIGHT	9	2
FRONTRIGHT	10	2
FRONTLEFT	11	2

Table 1
Old Sonar Mnemonics

In general, the functionality of the sonar control code remained the same in MML 11. However, function names differed slightly to adhere to the new function naming convention. The new convention eliminated the use of the underscore character (_) to separate words within a function name. Instead MML 11 used a combination of upper- and lower-case letters. Initially, sonar mnemonics also remained the same. For example, under MML 11, user program code may have contained the following:

```
EnableSonar(FRONTR);        /* Turn on sonar 3 */
DisableSonar(FRONTL);       /* Turn off sonar 0 */
LogSonar Data(FRONTR);      /* Begin logging data from sonar 3 */
```

In addition, Lieutenant Commander Kelbe converted the sonar data logging functions in MML 11 to reuse code from the motion data logging functions previously implemented.

3. Acoustic Characteristics

The angular beam pattern of an ultrasonic sensor is a critical parameter. Sherfey reported that the beam width of the major lobe was determined by using the accepted far-field approximation of

$$\theta = 1.22 \frac{\lambda}{D}$$

(Eq. 3-3)

where λ is the wavelength, D is the diameter of the uniform circular aperture and θ is the beam width in radians. Sherfey reported that for a 40 kHz signal and an ultrasonic sensor diameter of 1.5 centimeters, the acoustic wave length was 8.5 millimeters and produced a theoretical beam width of 40°. In hopes of reducing this beam width, the original design placed cones around the transducers as shown in Figure 6. (Sherfey, 1991, p. 51) From this geometry, Sherfey reported that the effective beam width at a distance of one meter was 2.6°. (Sherfey, p. 53.)

Additionally, Byrne conducted experiments in 1993 to investigated the data returned from the sonar system. He determined that Yamabico-11 could map out a straight wall accurately while performing either translational or rotational movement, but that the data returned from corners or round objects was sketchy and erroneous. (Byrne, pp. 29-43) In particular, Byrne observed that Yamabico-11 could not map either a concave or a convex 90 degree angle accurately.

Yamabico-11 plots concave right angle walls as a series of short askew line segments rather than one continuous line segment. The linear-fitting technique breaks down because of sides lobes which cause improper distances to be measured. Additionally, Yamabico-11 plots a line of sonar returns tangent to the vertex of the concave right angle. For the convex right angle, Yamabico-11 fails to get usable sonar returns. These problems will be explained in Chapter IV and are not a failure of Yamabico-11, but a limitation imposed by the physical nature of the problem.

Transmitter

23.5 mm

45 mm

$\theta = 1.3$

Receiver

47.5 mm

Sensor
Midpoint

1 meter
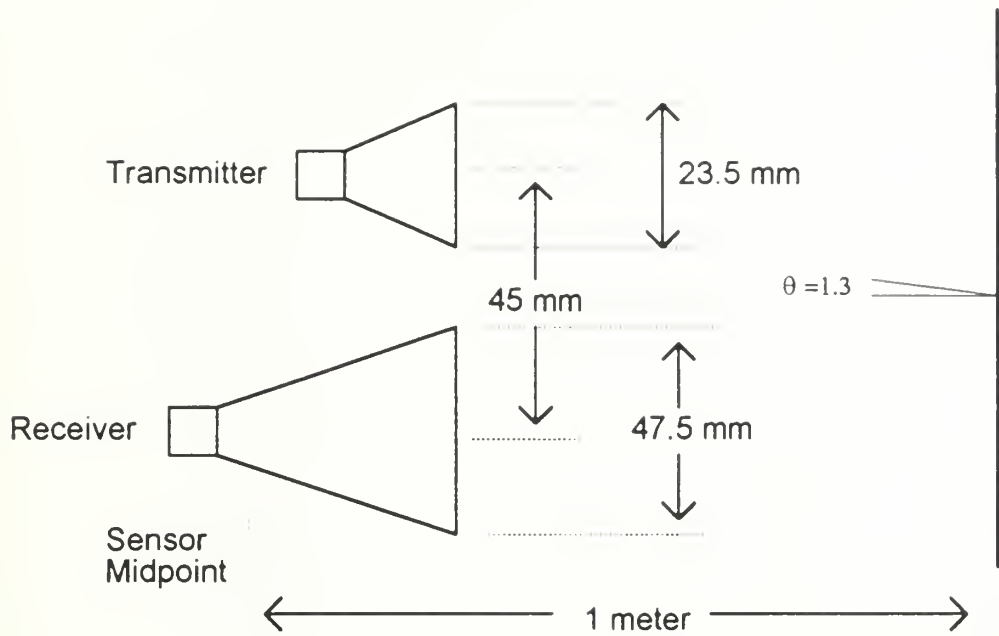
Figure 6
Sonar Cones and Effective Beam Width Calculation

IV. ACOUSTIC THEORY

A. BEAM WIDTH

The term "beam width" is used to describe the pressure field radiated by an energy source. It refers to the extremity of the major lobe of the radiation pattern. Sound levels received off-axis are weaker than those received on the centerline of the beam. Beam width can refer to one of several angles; the user must know which angle is being referenced. The ratio,

$$\frac{P(\theta)}{P_{ax}}$$

(Eq. 4-1)

where $P(\theta)$ refers to the off-axis pressure and P_{ax} refers to the centerline or axial pressure, is used to define the beam width angle. The common angles used are the half-power beam width, the half-amplitude beam width, and the nodal beam width. The half-power beam width, also referred to as -3 dB beam width, is the angle at which this ratio equals 0.707; the half-amplitude, or -6 dB, beam width occurs when the ratio is 0.5; and the nodal beam width takes place when the ratio goes to zero. Figure 7 shows the relationship of these different beam widths for an acoustically simple source with a circular aperture. Therefore, one must indicate at which point a given beam width was taken.

Determining the beam patterns of acoustic sources is greatly simplified if they can be treated as simple sources. "A *simple source* is a closed surface, vibrating with arbitrary velocity distribution, but of such a size that all dimensions are much smaller than the wavelength of the emitted sound (Kinsler, 1982, p. 164)." This is not the case for the sensors on Yamabico-11. The wavelength of the sound and the diameter of the transducer are almost equivalent. Hence, the transducer is a complex source.

Equation 3-3 represents the Fraunhofer diffraction of light through a circular aperture and represents the half angle from the central peak to the first node. It assumes that the angle subtended is small, using the small angle approximation of θ. If these angles are not small, then the correct expression

$$\sin \theta = 1.22 \frac{\lambda}{D}$$

(Eq. 4-2)

21
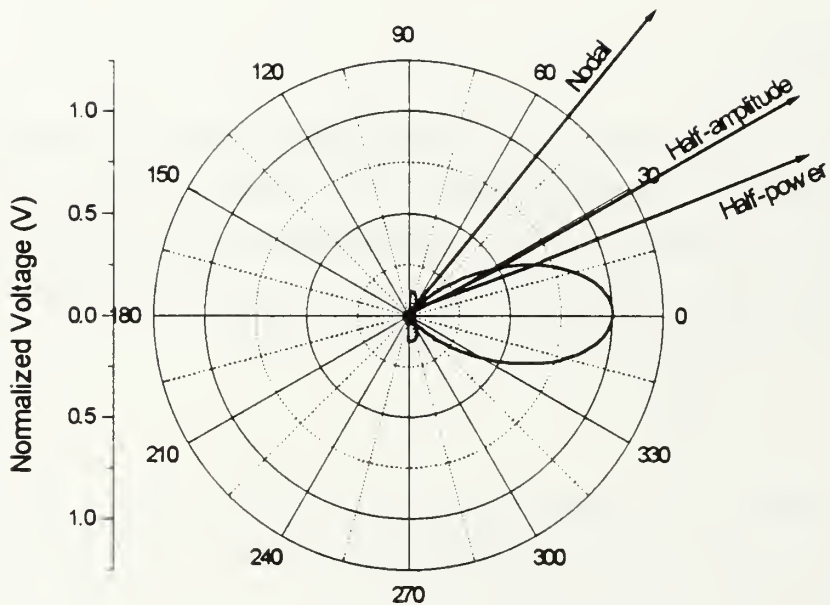
Figure 7
Common Beam Widths

must be used. Equation 3-3 is an excellent approximation in optics because the wavelength of light, λ, is usually much smaller than the diameter of the aperture ($\lambda \ll D$). It also assumes that the observation point is at a sufficient distance from the circular aperture that the light can be approximated by plane waves. The diffraction pattern produced is a central disk, known as the Airy disk, surrounded by concentric rings. The half angle subtended by the Airy disk is the angle θ in Equation 3-3. This is the half-angle of the first node.

However, in acoustics, λ is often of the same order of magnitude as D. Therefore, the angles are not necessarily small and substituting θ for $sin\theta$ is an invalid approximation; hence, Equation 4-2 is the correct equation. The sensors under investigation have an aperture diameter, D = 7.0 mm and acoustic wavelength, λ = 8.575 mm. Using the optical approximation, Equation 3-3 indicates that a node occurs at about 85.6 degrees. Equation 4-2 gives a non-real solution of $sin\theta$ = 1.4945, indicating that there is no nodal surface.

22

The initial assumption is that the source acts like a uniform circular piston mounted in a rigid baffle. The beam pattern function for this source is given by

$$P(\theta) = P_{ax}(\theta)\left[\frac{2J_1(ka\sin\theta)}{ka\sin\theta}\right] \qquad \text{(Eq. 4-3)}$$

where k is the wave number, a is the radius of the source aperture, J_1 is the first-order Bessel function and θ is the angle measured relative to the piston axis. The wave number $k = 2\pi f/c$ where f is the frequency and c is the speed of sound. Figure 8 plots the functional behavior of $2J_1(x)/x$. The sound pressure goes to zero when the plot in Figure 8 crosses the x-axis. This occurs when

$$ka\sin\theta_m = j_{1m} \qquad \text{(Eq. 4-4)}$$

where θ_m is off-axis angle of the node, and $j_{11} = 3.83$ is the value where the J_1 Bessel function goes to zero. For a frequency of $f = 40$ kHz, speed of sound in air of $c = 343$ m/s and aperture radius $a = 3.5$ mm, the term, ka, equals 2.565 and the first zero of the Bessel Function occurs at $\sin\theta_1 = 1.4945$. Solving for θ_1 gives a non-real solution, indicating that no nodal surface occurs within the 180 degree span. This result agrees with Olson who stated that a nodal surface will first appear for plane circular piston sources when the ratio of aperture diameter to wavelength of sound, D/λ, is greater than or equal to 1.25 (Olson, 1947, p. 39). For the transducer under investigation, $D/\lambda = 0.816$; therefore, no nodal surface was expected.

The sound pressure ratio from Equation 4-1 is the same as the ratio $2J_1(x)/x$; from Figure 8, the ratio $2J_1(x)/x$ equals 0.5 when $x \approx 2.2$. Since $x = ka\sin\theta$, this ratio occurs when $\theta \approx 59$ degrees, which once again agrees with Olson. The full beam width is twice this angle or 118 degrees. Since the technical data supplied with the transducers lists the half-amplitude beam width as 50 degrees, the assumption that the source alone acts like a plane piston is invalid.

A more probable assumption is that the source acts like a spherical source and the casing acts as a tube to produce a plane wave at the output of the sensor casing, imitating a circular piston source with radius equal to that of the case opening. The analytical solution to this problem is very complex due to the diffraction of the beam around the edges of the tube. In an infinite baffle, the
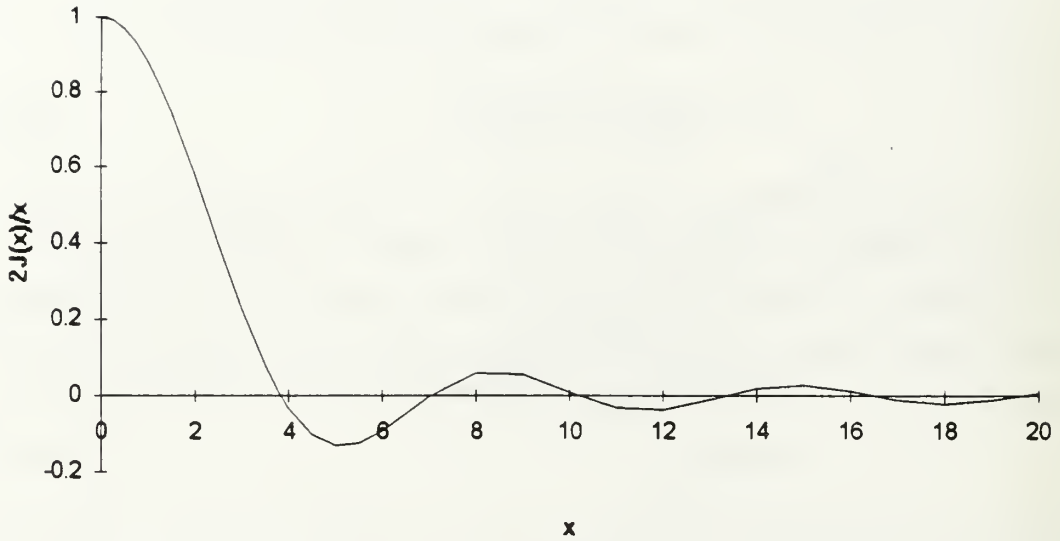
Figure 8
Behavior of the Function $2J_1(x)/x$

sound energy is reflected back into the beam; no energy can escape to the backside of the radiator. With the circular piston source in the end of a tube, the energy almost uniformly spreads 360° for a small *ka*. As the value of *ka* increases, the pattern produced has a strong, wide, irregularly-shaped beam in front of the radiator and a smaller, weaker, and still irregular beam pointing backward from the radiator.

Since the actual situation is difficult to solve analytically, the infinite baffle assumption was used with new aperture diameter equal to the inside diameter of the casing, 13.0 mm. Using 13.0 mm as the aperture diameter and Equation 4-2 yields a nodal half-angle of

$$\theta = \sin^{-1}\left(1.22\frac{\lambda}{D}\right) = 53.6 \text{ degrees} \qquad\qquad \text{(Eq. 4-5)}$$

Using the new aperture radius of 6.5 mm, the half-amplitude half-angle becomes $\theta = 27.51$ degrees, making the half-amplitude full beam width, $2\theta = 55$ degrees. This value is only 10% greater than that supplied in the technical data. Figure 9 plots this theoretical beam pattern.

24
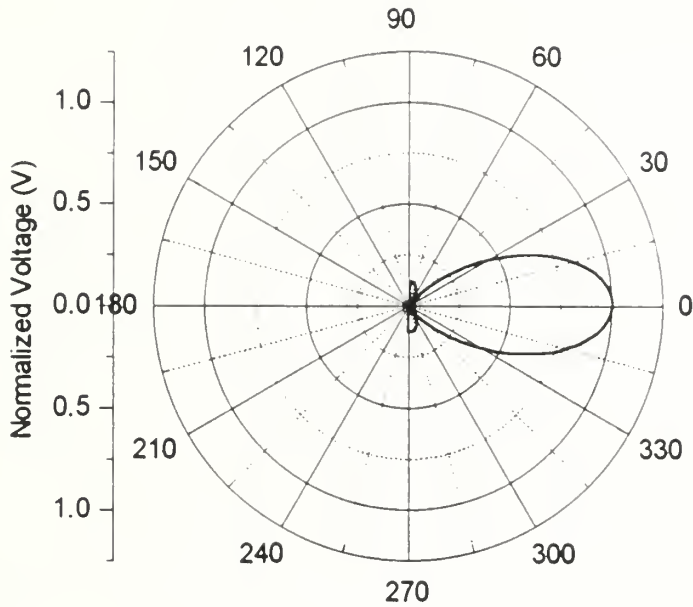
Figure 9
Theoretical Beam Pattern of Transducer with a = 6.5 mm and λ = 8.575 mm

B. HORNS

The main purpose of a horn is to increase the acoustical output of a piston-like transducer at low frequencies; increased directionality is a by-product. The horn acts as an acoustical transformer; it matches the impedance of air to that of the piston. At low frequencies, the acoustical impedance at the throat of the horn is greater than that which would act on a piston of equal area vibrating in an infinite baffle, resulting in a greater acoustical output. At high frequencies, the horn has little effect. At high frequencies the transmitted beam is much narrower; the horn does not increase the acoustical impedance.

If the wavelength of sound is greater than the diameter of the horn mouth, then the directional characteristics will be determined by the mouth; otherwise, it is the flare of the horn which determines its directional characteristics. At high frequencies, the wavelength is small so flare is important. The most effective horn is one in which the rate of flare increases from throat to mouth.

Hyperbolas, catenaries and exponential functions have all been used to determine the flare rate. The most commonly used horn employs the exponential function

$$S_x = S_0 e^{mx}$$
(Eq. 4-6)

where S_x is the cross-sectional area at any given position x, S_0 is the cross-sectional area of the throat given by $S_0 = \pi r^2$ where r is the physical radius of the transducer element, and m is the flare constant. (Kinsler, 1982, p.373) However, the horns used on Yamabico-11 are of the simpler conical shape where

$$S_x = S_0 x^2$$
(Eq. 4-7)

The size of the conical horn is important in determining the beam characteristics it will produce. At low frequencies, where the wavelength of sound is greater than the diameter of the horn mouth, the pattern is the same as that produced by a piston of the same size as the mouth of the horn. The acoustic waves exiting the mouth are essentially planar. At higher frequencies, the pattern becomes narrower until it crosses over a threshold, at which point the exiting acoustic waves are no longer planar. At even higher frequencies, the circular conical horn acts the same as a spherical surface source whose radius is equal to the distance as measured along the side of the horn from the imaginary apex to the mouth opening. The exiting acoustic waves are now spherical. As the frequency continues to increase, the pattern begins to broaden out again. (Olson, pp. 42-43)

Yamabico-11 uses two different sized conical cones with the ultrasonic transducers mounted inside. Both cones have a radius of 9.25 mm at the point where the transducers are mounted; this is the throat radius. The small cone on the transmitter has an angle of opening, θ_s, of 23.7°, with a mouth radius of 13.25 millimeters or about 1.56λ, and overall length of 63.20 millimeters or about 7.4 wavelengths from the transducer. The larger cone on the receiver has a θ_s of 28°, a mouth radius of 23.75 millimeters which is about 2.8λ and overall length of 95.26 millimeters or about 11.2 wavelengths from the transducer.

As has been stated previously, the acoustic waves exiting the bare transducer are planar; therefore, the bare transducer is already producing the desired beam pattern. The addition of a cone introduces impedance mismatches at both the throat and mouth of the cone. These mismatches cause a reflected wave back towards the transducer, setting up interference. Hence, the cones do not enhance the acoustical properties of the transducers.

C. REFLECTION

Using only a sonar system, a robot is unable to map convex and concave right angles accurately. The explanation for this can be found in basic physics principles. Detecting a convex right angle is virtually impossible given the configuration in Figure 10. The transmitted sound is reflected off the wall, away

Wall or Object

Acoustic Signal

Robot's path
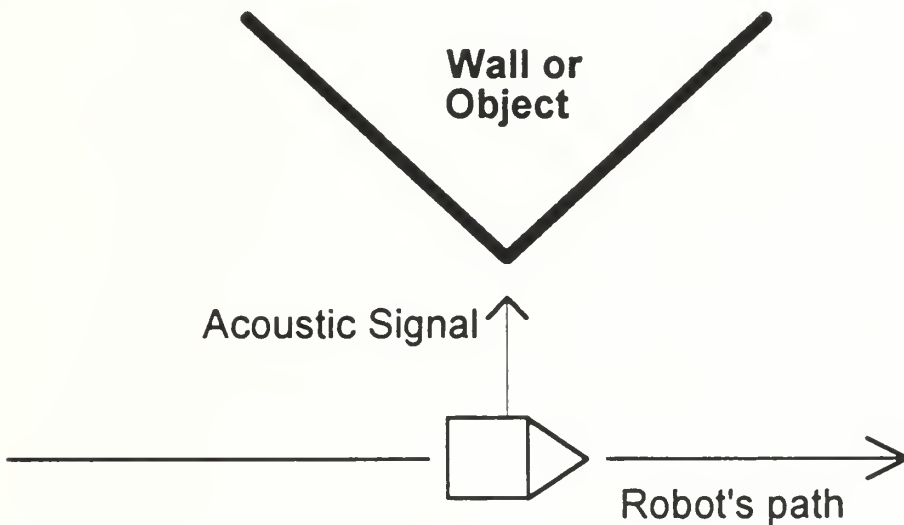
Figure 10
Convex Right Angle Detection and Mapping

from the robot. The angle of reflection, θ_r, equals the angle of incidence, θ_i. In Figure 5, $\theta_i = 45°$; therefore, the signal will be reflected off of the wall at 45°. When the robot is directly abreast of the corner as in Figure 10, some of the signal will be reflected from the very tip of the corner, but this reflected signal is very weak. Only a very small percentage, less than one percent, of the transmitted beam impinges on the corner tip in a manner so it may reflect back

to the robot. Since the percentage of signal returned is so low, it is then lost in the circuit noise.

The concave right angle is easier to detect although the physical nature of the problem makes it difficult to map the concave right angle accurately. The distances measured in the vicinity of the angle are longer than they actually are. Figure 11 shows a line segment extracted from the sonar return in the vicinity of the concave right angle. As expected, the line segment is tangent to the vertex.

Acoustic Signal

Robot's path

Legend: Distances as mapped by Yamabico-11
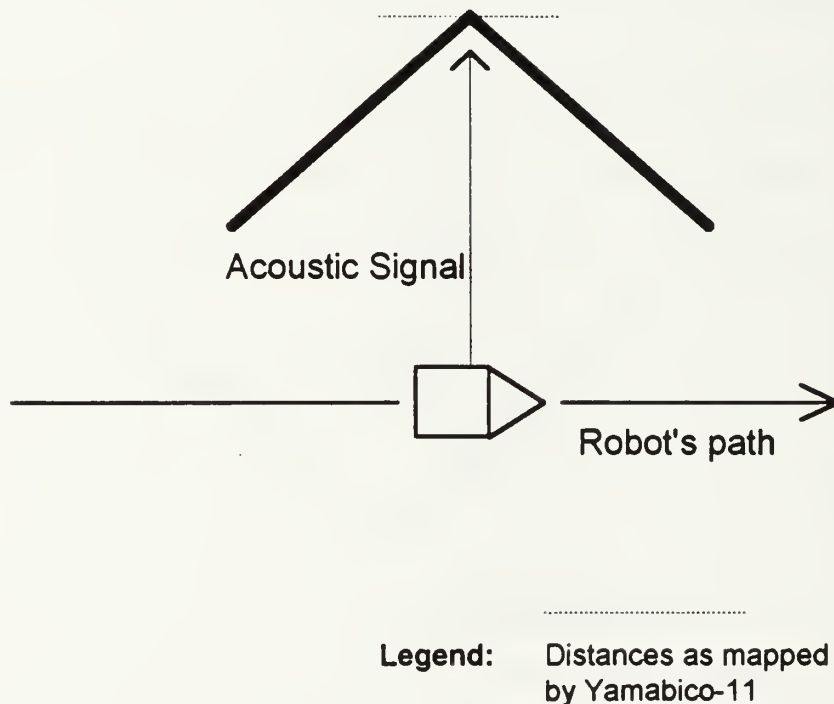
Figure 11
Concave Right Angle Detection and Mapping

A concave right angle acts as a retro-reflector; the reflected signal is parallel to the transmitted signal. The path length (time) measured by the sonar system is longer than the actual distance. Figure 12 blows up the corner and shows what is happening. The distance plotted is based on the total path length vice the actual distance to the wall. The distance plotted is

$$x = \frac{1}{2}(d1 + d2 + d3)$$ (Eq. 4-8)

As long as the receiver is within the reflected beam, it will measure this distance. The size of the beam when it reaches the wall depends on the beam width and distance to the wall. The beam radius is

$$r = d \tan \theta \qquad \text{(Eq. 4-9)}$$

where d is the distance to the wall and θ is the half-amplitude beam width. Using the half-amplitude beam width of 27.51° calculated for the sensors used on Yamabico-11 in Section A above, the diameter of the reflected signal is about equal to the distance to the wall. For all practical purposes, the receiver will be within the reflected beam.

x

θ1 θ2

d2

θ1 θ2

d1 d3

Transmitter Receiver

x
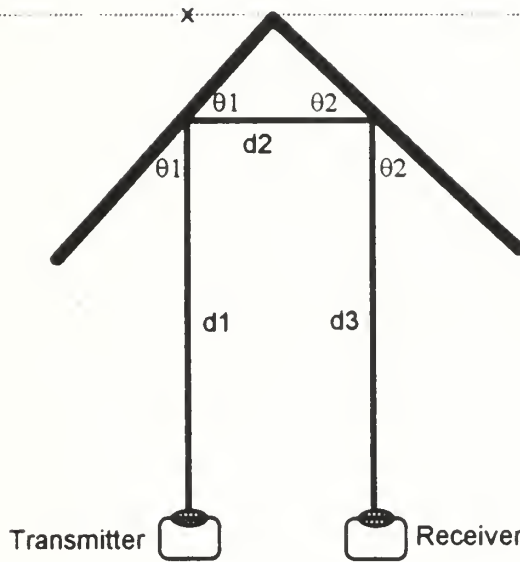
Legend: Distances as mapped
by Yamabico-11

Figure 12
Reflection Plotted by Yamabico-11

Although the convex right angle can not be mapped given the situation depicted in Figure 10, the concave right angle can be mapped accurately under the situation in Figure 11. Although Figure 12 shows $\theta 1 = \theta 2 = 45°$, the same line

would occur for $\theta 1 \neq \theta 2$. However, the corrections needed rely on the fact that the robot *knows* that it is encountering a concave right angle. This type of knowledge would only exist if the robot already has a map of its world and is just verifying its position within that map. The corrections could *not* be applied to sonar returns resulting from the detection of an unknown object.

V. ANECHOIC CHAMBER EXPERIMENTS

A. BEAM PATTERNS

The first experiment conducted was to verify the beam pattern of the Nicera transducer elements. Figure 13 shows the experiment setup used to verify the beam width. The technical data indicates that this sensor operates at a frequency of 40 kHz, has a –6dB half-amplitude beam width of 50°, and can
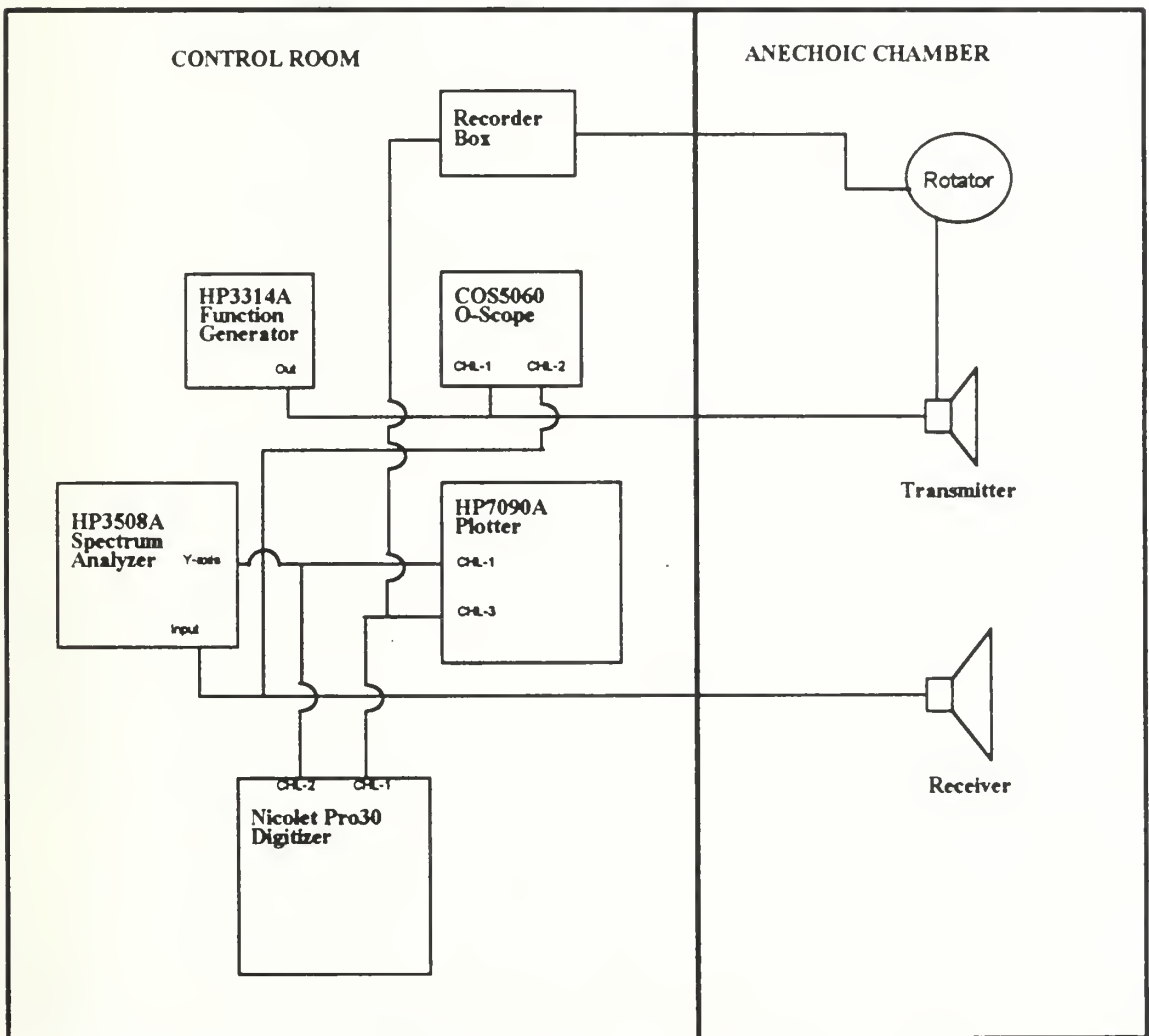
Figure 13
Beam Pattern Experiment Setup

31

handle inputs up to 20 Vrms. The transmitter and receiver were mounted in a manner to ensure that they were operating in the far-field. Figure 14 shows the actual setup in the anechoic chamber used for this first experiment.
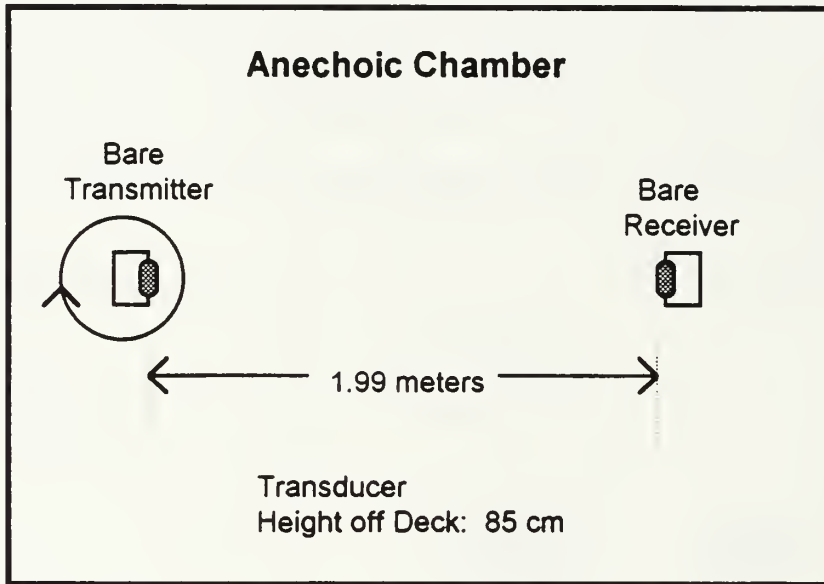
Figure 14
Experiment 1 Setup in the Anechoic Chamber

The voltage used to conduct these experiments was 4.5 Volts peak-to-peak as this was the effective voltage used by Yamabico-11. This preliminary experiment was necessary to validate the experimental procedures which would be used later on. Ideally, the experiment would have been conducted using an omnidirectional transmitter and receiver to measure the response of the subject receiver and transmitter, respectively. However, an omnidirectional receiver or transmitter which operated at 40 kHz was not available. Therefore, the experiment would have to be conducted using an "identical" transmitter and receiver.

Figure 15 shows the linear data extracted during this experiment. Figure 16 depicts this information in polar form. The theory presented in Chapter IV had predicted that the node would occur at 52.9°. Experimentally, the node was measured at about 58°; a difference of less than 10%. The -6dB value for the half-amplitude comes from the equation

32
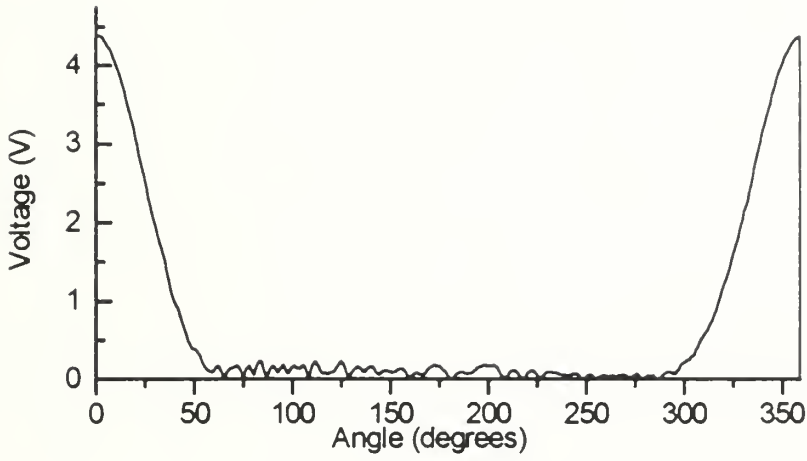
Figure 15
Measured Results of Bare Transmitter/Receiver
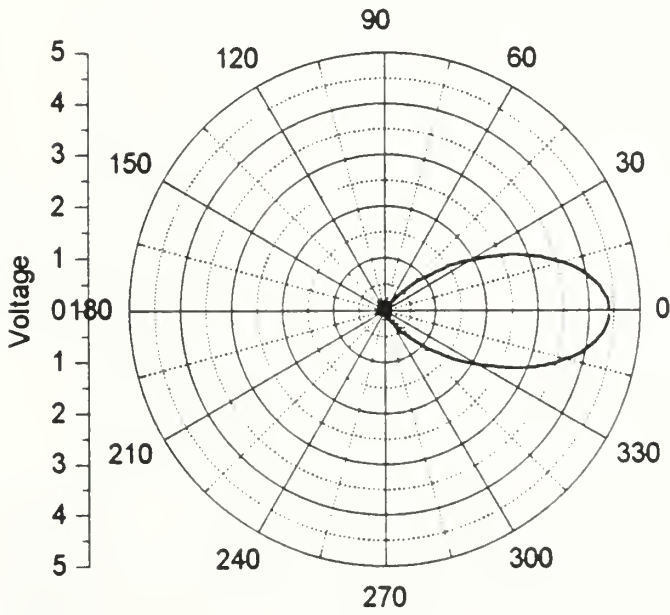
Figure 16
Bare Transmitter/Receiver Beam Pattern

$$20\log\left(\frac{P(\theta)}{P_{ax}(\theta)}\right) = -6 \qquad\qquad \text{(Eq. 5-1)}$$

where the quantity $P(\theta)/P_{ax}(\theta) = 0.5$, the half-amplitude. At half amplitude, θ' was 29° ± 1°. A theoretical θ' of 27.51° calculated in Chapter IV. Therefore the -6dB beam width was 2 x θ' or 58° which is 5.4% greater than the theoretical half-amplitude beam width and 16% more than that reported in the technical data accompanying the sensors. All comparisons hence forth will use the half-amplitude as the threshold of an acceptable return. The actual data extracted during the experiment is located in Appendix A.

Next, the effects of the transmitter and receiver cones were examined. The transmitter and cone combination was attached to the rotator and the bare receiver was placed approximately two meters away as per Figure 17.
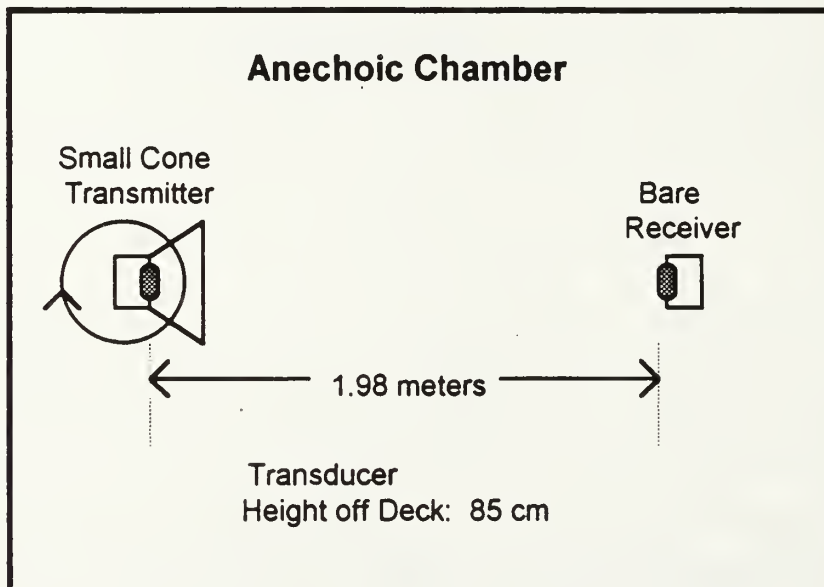
Figure 17
Experiment 2 Setup in the Anechoic Chamber

The results plotted in Figures 18a and 18b indicate that the small transmitter cone was detrimental to the signal. Although the beam width of the major lobe was reduced, adding the small transmitter cone created side lobes.
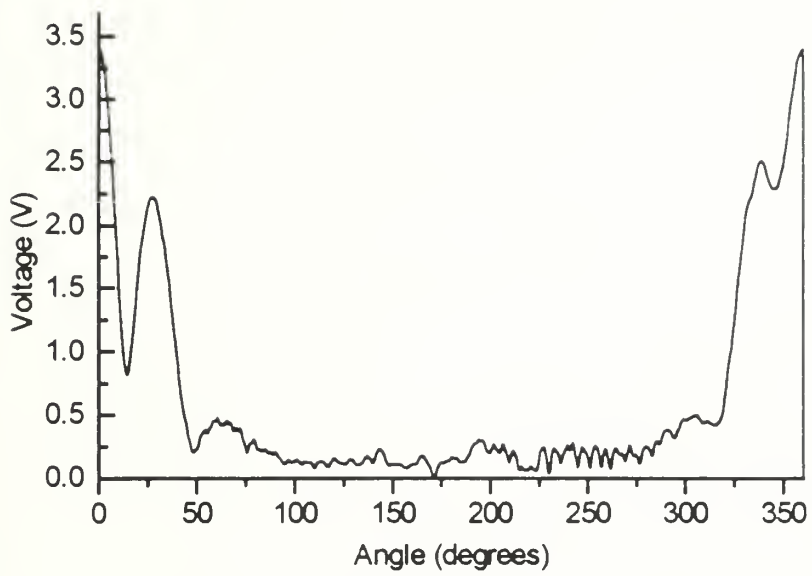
Figure 18a
Linear Plot of Beam Pattern Produced by Small Transmitter Cone
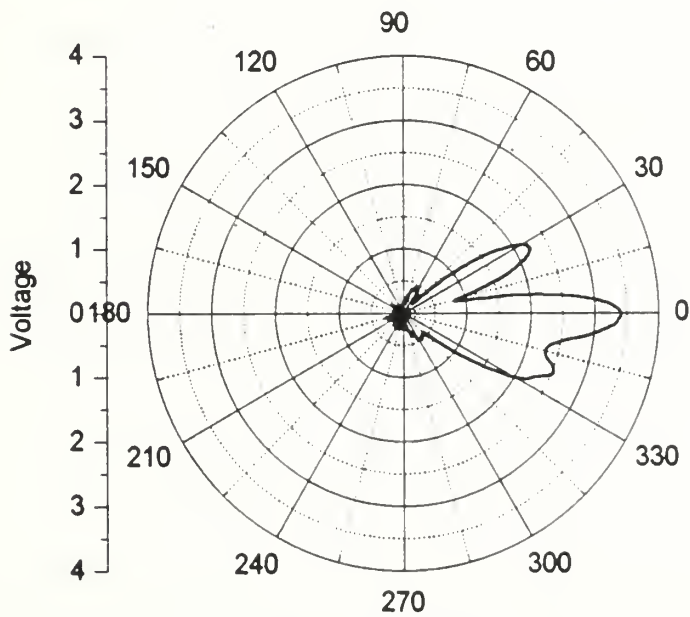
Figure 18b
Polar Plot of Beam Pattern Produced by Small Transmitter Cone

These side lobes are undesirable because they can lead to false readings. Appendix A contains the analog graphs obtained during this experiment.

The experiment was then reversed; the bare transmitter was fixed at a distance of about two meters and the receiver and cone combination was rotated as per Figure 19.

Anechoic Chamber

Large Cone
Receiver

Bare
Transmitter

←———— 1.86 meters ————→

Transducer
Height off Deck: 85 cm
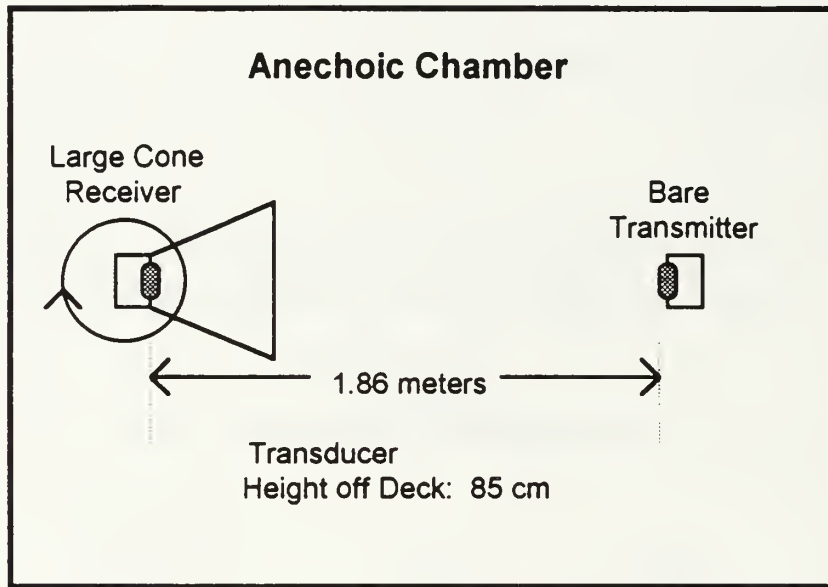
Figure 19
Experiment 3 Setup in Anechoic Chamber

Figures 20a and 20b shows the affects of the larger cone on the receiver. As before, the reduction in the major lobe beam width came at the expense of more side lobes. The analog graphs recorded during the experiment are contained in Appendix A.

B. DETECTION CHARACTERISTICS OF SONAR PAIR

Sherfey concluded that the current sonar configuration was sensitive to the orientation of the reflecting surface. He stated that the sonars had to be within a few degrees of perpendicular to the surface in order to get a return, thereby giving good bearing resolution. (Sherfey, 1991, p. 53) To investigate the characteristics of the sonar pair, a wall was built in the anechoic chamber. This
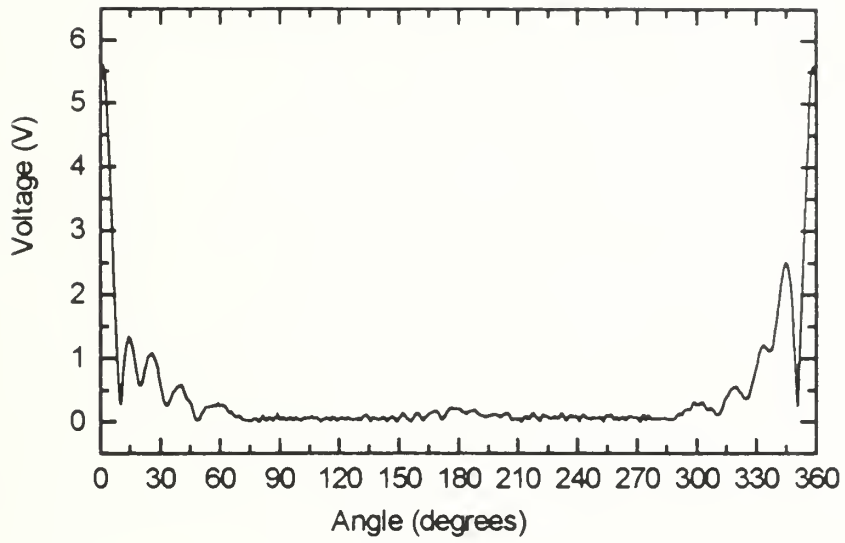
Figure 20a
Linear Plot of Beam Pattern Produced by Large Receiver Cone
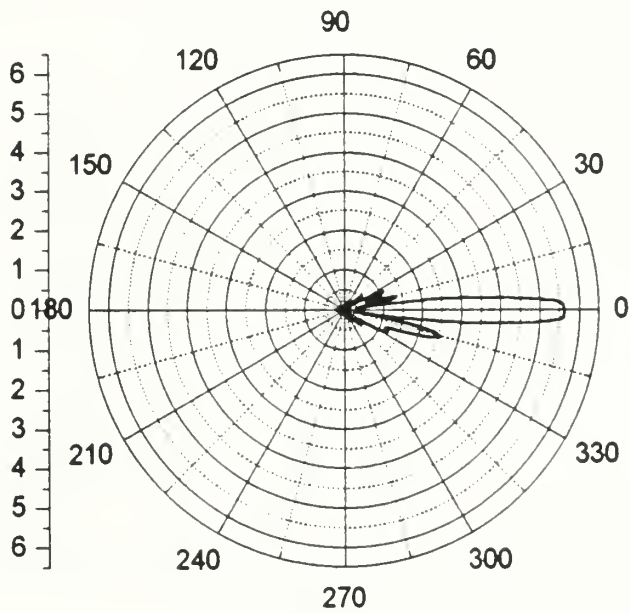
Figure 20b
Polar Plot of Beam Pattern Produced by Large Receiver Cone

37

wall was not a perfect reflector; however, any sound energy that was transmitted through the wall material or absorbed by the wall material could be ignored since the experiment was concerned with relative, vice absolute, measurements of amplitude and the sound loss was a constant. The wall was placed within the far-field, but close enough to the sensor pairs to allow almost the entire transmitted beam to ensonify the wall. The series of experiments tested both the current coned sonar configuration and for the bare sonar configuration. For both configurations, the experiments collected data with the sonar pair in both the horizontal and vertical positions and rotated the sonar pair in both the clockwise and counter-clockwise directions. Neither the sensor orientation plane nor rotation direction affected the outcome. Figures 21a and 21b show the experimental setup and the polar graph of the received signal for the current cone configuration. Figures 22a and 22b show the experimental setup and the polar graph of the received signal for the bare transducer configuration.
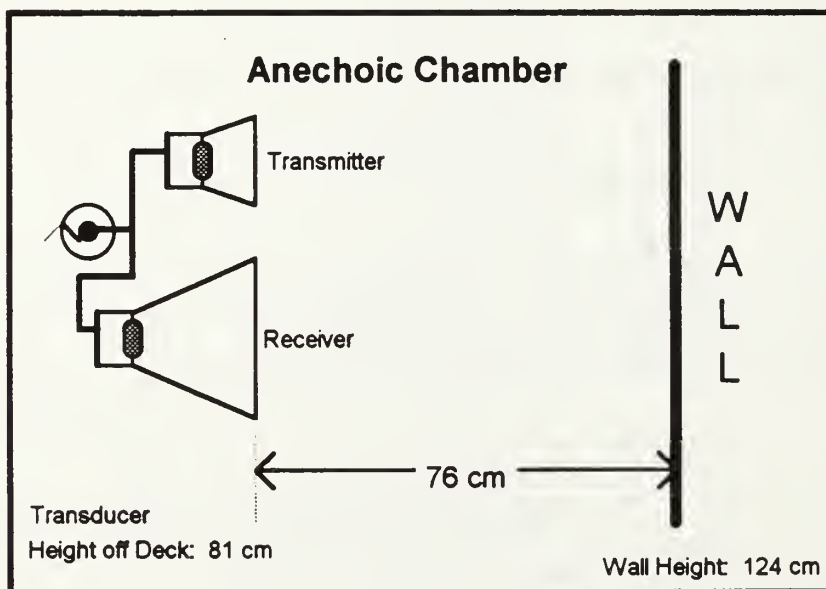
Figure 21a
Experiment 4 Setup in the Anechoic Chamber
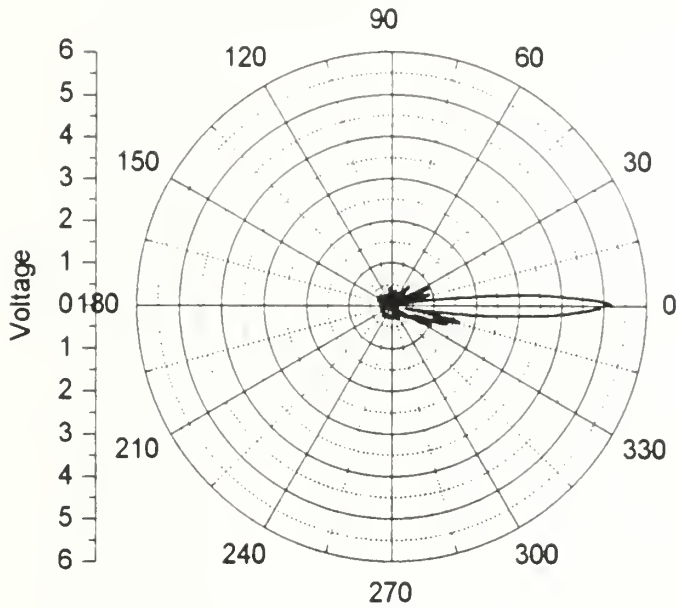
Figure 21b
Cone Configuration Angular Sensitivity
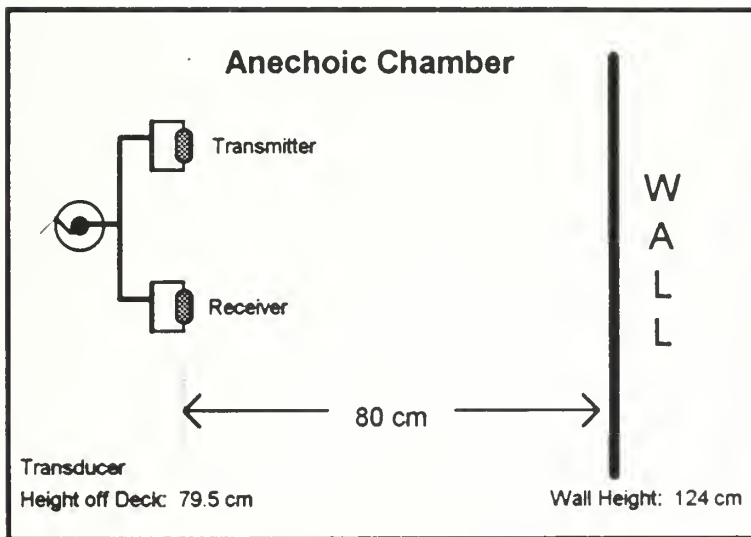
Figure 22a
Experiment 5 Setup in the Anechoic Chamber
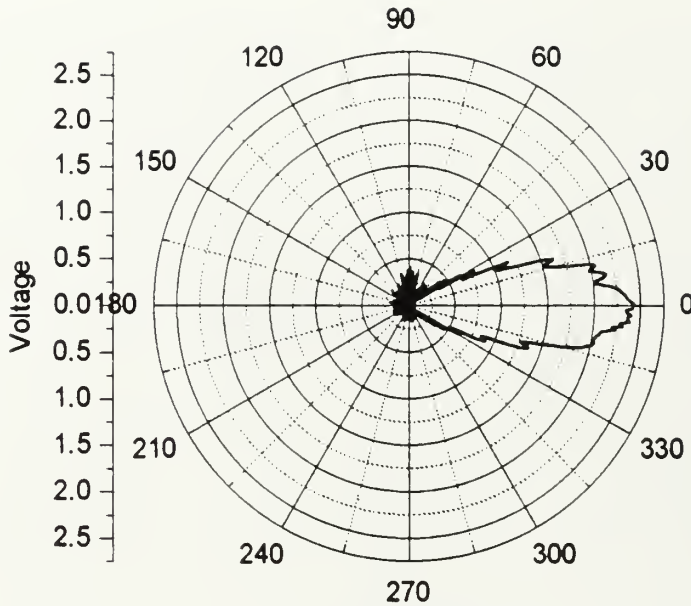
Figure 22b
Bare Transducer Angular Sensitivity

As expected, in both cases, the beam pattern produced resulted from the multiplication of the individual transmitter and receiver patterns. Sensor orientation, horizontal vs. vertical, did not affect the resultant pattern; however, the cones had a dramatic affect. Using the half-amplitude beam width for comparison, the current cone configuration requires that the sensor pair be within about 5° of normal in order to detect an object. If a bare sonar pair is used, a usable return up to 25° off normal is possible, as can be seen in Figure 22b. The actual analog data recorded during these experiments are contained in Appendix A.

VI. YAMABICO-11 EXPERIMENTS

A. DISTANCE MEASUREMENTS

1. MML 10

Since MML 11 was still under development when this research began, MML 10 and the Motorola microprocessor were used to verify the sonar system's distance calculation using both the old, coned sonar pair configuration and the new, bare sonar pair configuration. For these experiments, a sonar pair without cones replaced the front left sonar pair, Sonar 0, on Yamabico-11. The bare sonar pair mounted its transmitter and receiver flush, keeping the separation distance as before at 45 millimeters. The 5th floor of Spanagel Hall at the Naval Postgraduate School served as the experimental laboratory. Marks on the floor indicated distances from the wall; these marks occurred at 10 centimeter increments up to one meter, then at 50 centimeter increments thereafter up to 400 centimeters, then at 410, 415 and 420 centimeters. These marks had an accuracy of ± 0.1 centimeters. Despite great care, aligning Yamabico-11 on the marks introduced another ±0.1 centimeter error. Positioning Yamabico-11's sonar beam perpendicular to the wall introduced an error which depended on the distance from the wall; at 400 centimeters, being 2 degrees off of perpendicular introduced a +0.24 centimeter error. Using both the new bare sonar pair configuration, Sonar 0, and the old coned sonar pair configuration, Sonar 3, the experiments recorded the distances measured by the sonar system. Each experimental run used the average of twenty-one readings. Comparison of the averaged raw range data points and the actual marked distance produced a difference, called "Delta," calculated by subtracting the distance determined by the sonar from the actual marked distance.

Figures 23 and 24 show the results of these experiments using the original circuitry. The slopes of the best fitting line to the data are within less than 0.5% of each other and the y-intercepts are within less than 4% of each other. There was less error in the distances as measured by Sonar 0; the standard error for Sonar 0 was 0.52 centimeters whereas Sonar 3 had a standard error of 0.77 centimeters. In both cases, zero error occurred at

Distance Error Bare Sonar Pair

Distance (cm)

Data Points .... Best Fit
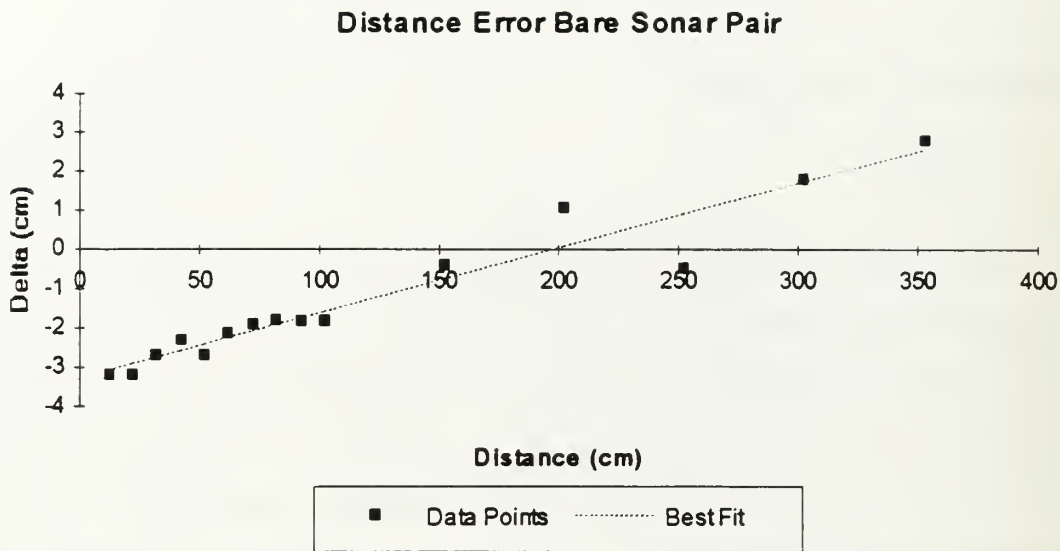
Figure 23
Error in Bare Sonar Pair Distance Measurement Using Old Circuitry
Equation of "Best Fit" Line: Y = 0.016581 X - 3.26489

Distance Error Coned Sonar Pair

Distance (cm)

Data Points .... Best Fit
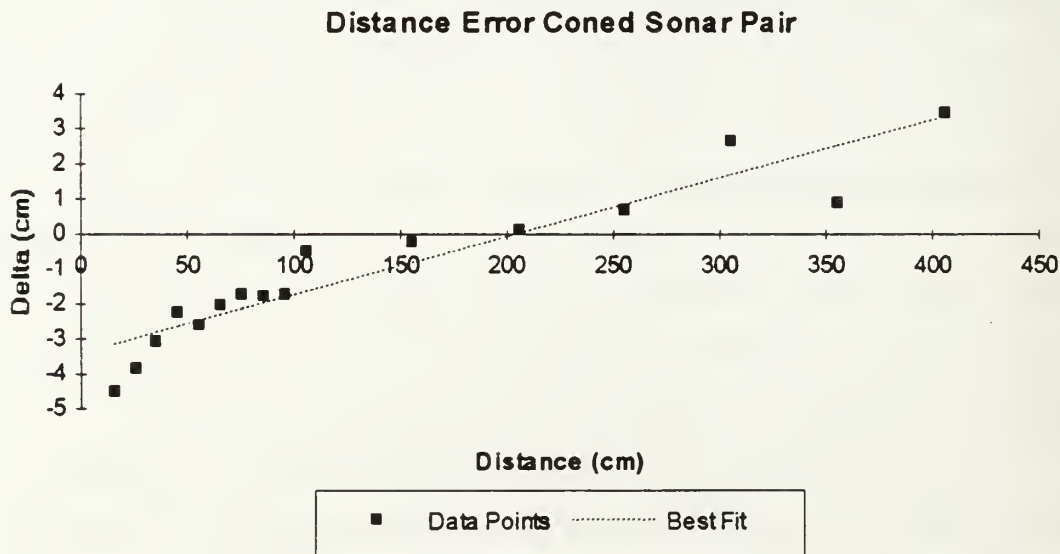
Figure 24
Error in Coned Sonar Pair Distance Measurement Using Old Circuitry
Equation of "Best Fit" Line: Y = 0.016653 X - 3.38523

approximately half the maximum distance. Although Sonar 3 was able to get a return at 4.053 meters, it failed to get a return at 4.153 meters. Sonar 0 failed to get a return at distances beyond 4 meters. The theoretical maximum range in both cases was 4.214 meters. Based on this information, it appeared that the sensor configurations had no affect on the effectiveness of the current distance calculation algorithm and that neither configuration was able to achieve the maximum theoretical range even under the ideal circumstances of this experiment.

Subsequently, similar experiments took data measurements at 50 centimeter increments using the new circuitry designed by Michiue. Figures 25 and 26 show the results of these experiments. With the new circuitry, the slope of the best fitting line to the data increased while the y-intercept remained about the same. But most importantly, with the new circuitry, both sonars were able to get consistent returns at over 4 meters. These experiments showed that the new circuit design had increased the range for expected returns. However, these experiments also demonstrated the inaccuracy of the current distance calculation algorithm.

Distance Error Bare Sonar Pair (New Circuit)
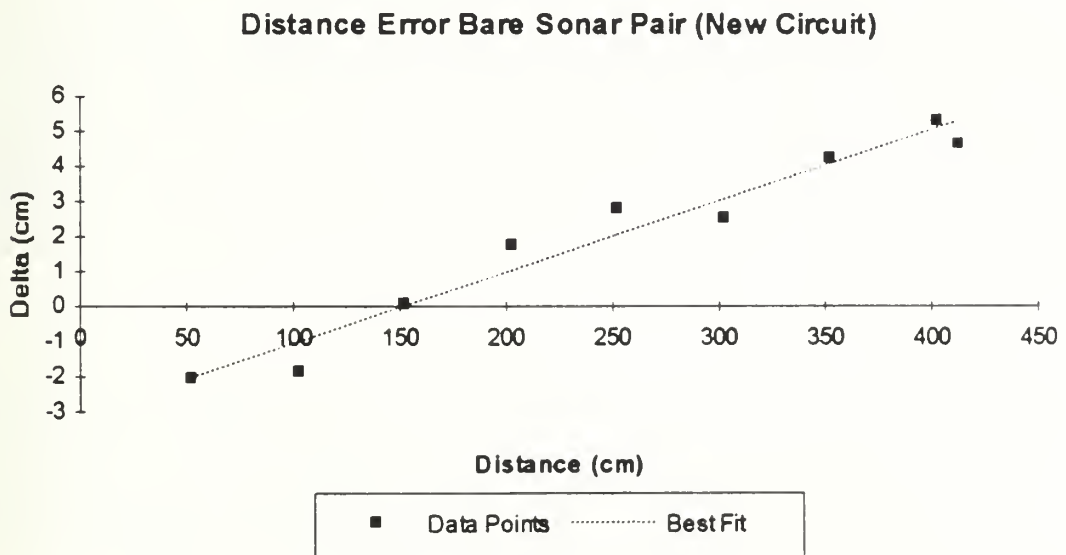
Figure 25
Error in Bare Sonar Pair Distance Measurement Using New Circuitry
Equation of "Best Fit" Line: Y = 0.020333 X - 3.07433

43

Distance Error Coned Sonar Pair (New Circuit)
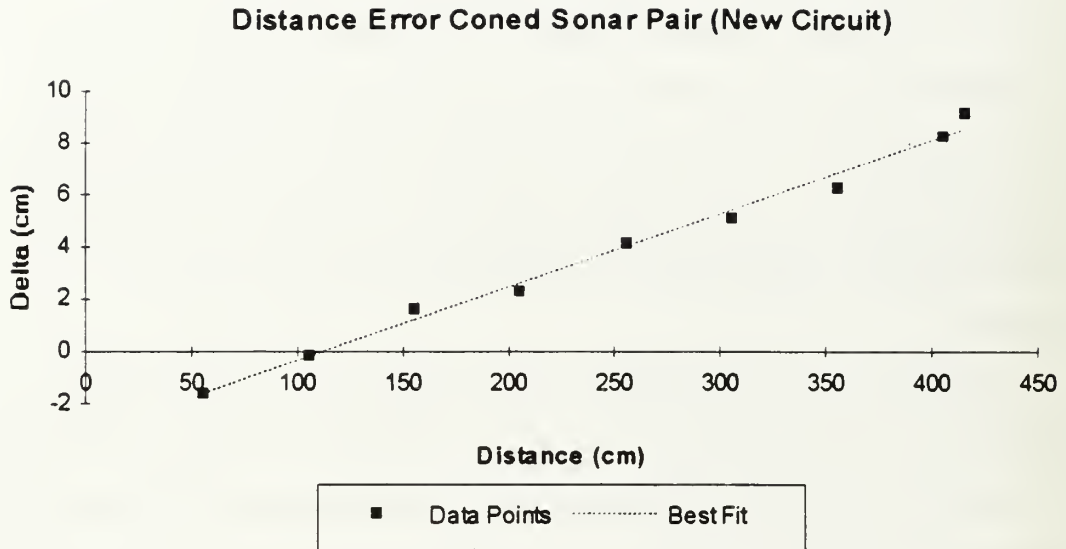
Figure 26
Error in Coned Sonar Pair Distance Measurement Using New Circuitry
Equation of "Best Fit" Line: Y = 0.028235 X - 3.16069

Ideally, the best fitting line to the data should have zero slope and fall on the x-axis. Investigation revealed that the slope and offset in the data results from hardware timing constraints. The function "serve_sonar" contained in the file "sonarcard.c" in MML 10 calculates the distance measured by the sonars. To determine the distance, a register records the number of clock ticks between the transmission of the pulse and the reception of the return pulse. The function reads this number from the register and divides it by ten, calling this the raw range. However, the distance traveled is

$$d = t^*c \qquad \text{(Eq. 6-1)}$$

where t is the time of one clock cycle and c is the nominal speed of sound in air. In 6 microseconds, sound travels 0.2058 centimeters at 343 m/s. Therefore, for two-way travel, the linear distance represented by one clock tick is one-half this value or 0.1029 centimeters vice 1/10 centimeter as previously coded.

44

2. MML 11

Implementation of the sonar functions into MML 11 completed concurrently with the above experiments in September 1994. Therefore, implementation of the correct clock-distance conversion factor occurred in MML 11 in the file "sonar.c" under the function "SonarSysControl." All subsequent experimentation used MML 11 and the SPARC4 microprocessor.

Although it seemed like a minor point, dividing the number of clock ticks by ten versus multiplying them by 0.1029 greatly affected the distance error. The previous experiments were repeated using the new clock-distance conversion factor; Figures 27 and 28 plot the results. In both cases, the slope went from positive to negative. Since "Delta" is the actual measured distance minus the sonar distance, this means that the sonar system is recording a longer distance which is consistent with the received pulse shape limitations discussed in Chapter III.
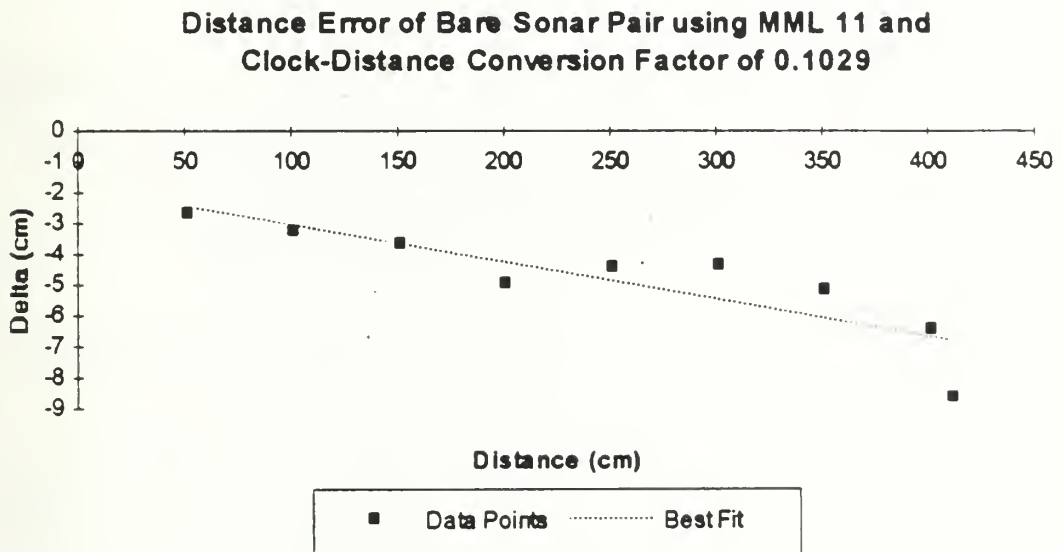
Figure 27
Bare Sonar Pair Distance Error
(Clock-Distance Conversion Factor of 0.1029)
Equation of "Best Fit" Line: $Y = -0.01214 X - 1.817$

**Distance Error of Coned Sonar Pair using MML 11 and
Clock-Distance Conversion Factor of 0.1029**
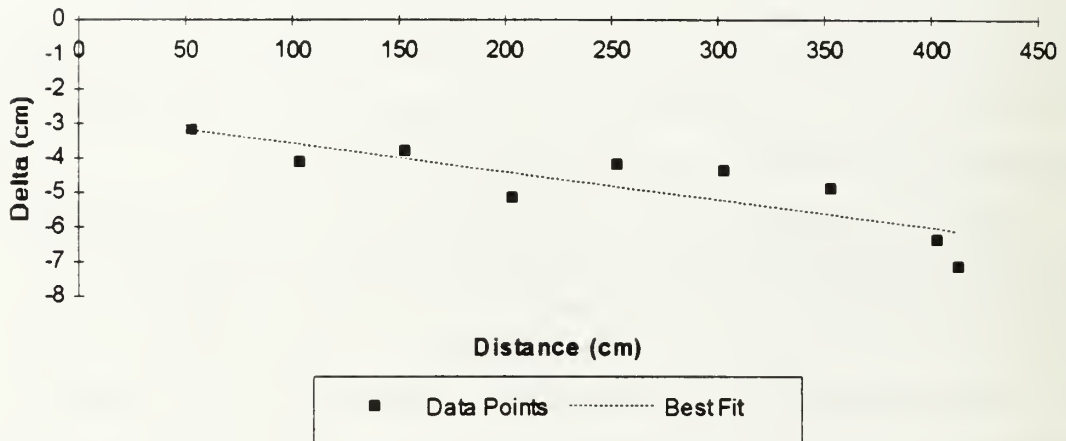
Figure 28
Coned Sonar Pair Distance Error
(Clock-Distance Conversion Factor of 0.1029)
Equation of "Best Fit" Line: Y = - 0.00815 X - 2.771

There are both static and dynamic causes of the error. The static causes related to the clock counter are minor; the main cause of error is dynamic and related to the strength of the reflected signal and the firing of the Schmitt Trigger. As previously stated, the strength of the received signal is a function of both the distance to and the reflectance of an object.

Additional experiments examined the amount of error introduced by both distance and object composition. Adjustment of the oscilloscope to show the individual cycles of the received signal and the firing of the Schmitt Trigger while the sonar pinged continually allowed counting of the number of cycles received before the Schmitt Trigger fired. For the bare sonar pair, Sonar 0, it took three cycles at 50 centimeters before the Schmitt Trigger fired; at 400 centimeters, it took about 13 cycles. For Sonar 3, the coned sonar pair, it took two cycles at 50 centimeters and eight cycles at 400 centimeters. Between these ranges, the number of cycles needed to fire the Schmitt Trigger rose linearly. Each cycle caused the sonar range to be about 0.43 centimeters greater than the actual distance. At longer distances, the number of cycles needed to fire the Schmitt

46

Trigger varied. The number recorded represented the average observed over a period of a couple of minutes.

To examine the affect of object composition, Yamabico-11 moved slowly , towards an object, stopping when the sonar range fell below 150 centimeters. Recording the actual distance from the various objects to the front of the stopped Yamabico-11 enabled a comparison of objects with different material composition. Although the value of actual measurement was not important, the difference between the measurements was significant. Table 2 lists some of these distances. Since the point at which the Schmitt Trigger fires varies, it is impossible to develop a software algorithm to calculate the distance accurately using the circuitry of Figure 4.

OBJECT ENSONIFIED	DISTANCE (± 0.5 cm)
Cardboard Box	137.0
Carpeted Room Divider	139.5
Foam Rubber	130.0
Plastic Trashcan	143.5
Wall	140.0

Table 2
Sonar Distance Variations Based on Material Composition

B. MULTIPATH INTERFERENCE

The anechoic chamber experiments showed that the bare sonar pair produced a wide beam pattern whereas the coned sonar pair produced a narrow beam with side lobes. Therefore, the next testing configuration placed the bare sonar pair, Sonar 0, above the coned sonar pair, Sonar 3, at a height of approximately 45.5 centimeters from the floor on robot centerline. This would allow usage of both wide and narrow beam sensors. Additionally, the front corner sonars, Sonar 10 and Sonar 11, became bare sonar pairs. Repeating the previously described distance experiments for both the bare and coned sonar pairs, Sonar 0 and Sonar 3 respectively, revealed problems. The coned

sonar pair performed consistently, but at approximately 3 meters, the return signal disappeared for bare sonar pair. Although both the corner sonars, Sonar 10 and Sonar 11, had the same bare sonar pair configuration, their return signals were consistent. The only difference between the three bare sonar pairs' was mounting height. The corner sonars were mounted at approximately 36.5 centimeters from the floor.

Although the bare sonar pair provided a significant increase in obstacle detection capability, it was susceptible to multipath interference. Figure 29 shows the multipath phenomena. If the path length difference, *2d-D,* in Figure 29 is equal to an odd number of half-wavelengths, then total destructive interference occurs

$$2d - D = \left(n + \frac{1}{2}\right)\lambda \qquad \text{(Eq. 6-2)}$$

where *n* is a positive integer and the wavelength of sound, λ = 8.575 millimeters.
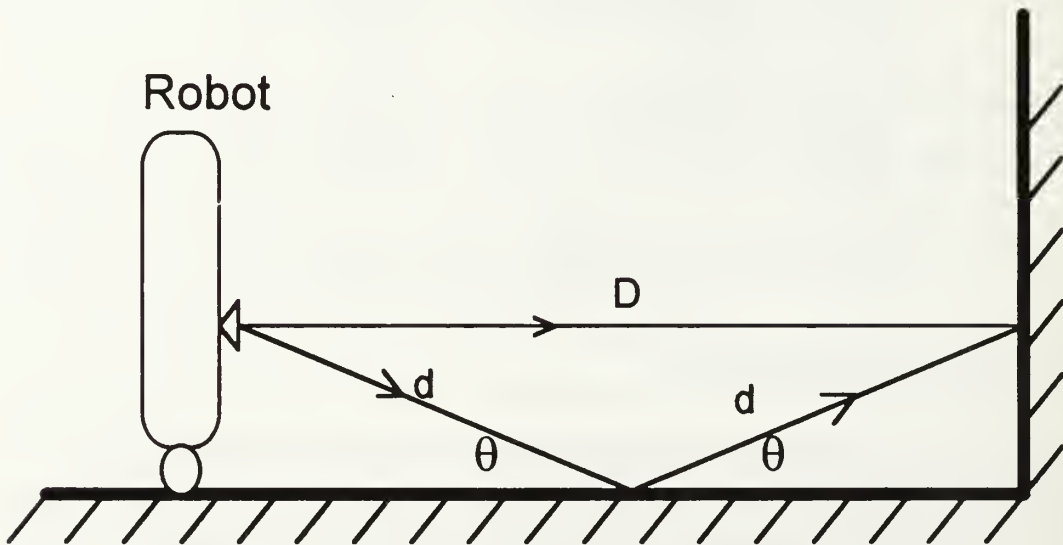
Figure 29
Multipath Interference Problem

Countering the multipath interference effect requires varying either the height, the beam width or the wavelength. The operating frequency fixes the wavelength so beam width and height are the only variables. The sensor height is given by

48

$$h = d \sin \theta \qquad\qquad\qquad\qquad \text{(Eq. 6-3)}$$

Combining Equations 6-2 and 6-3 means that total destructive interference will occur if

$$\theta \ge \sin^{-1}\left(\frac{2h}{\left(n + \frac{1}{2}\right)\lambda + D} \right) \qquad\qquad \text{(Eq. 6-4)}$$

At a height of 36.5 centimeters and distance of 4.214 meters, total destructive interference will occur is $\theta \ge 9.96°$. For a distance of 2 meters, a sensor at this same height will experience total destructive interference if $\theta \ge 21.35°$. However, the thresholding caused by the Schmitt Trigger means that θ is a function of reflected signal strength and difficult to predict. This makes it virtually impossible to calculate the optimum height to avoid total destructive interference within the operating range of Yamabico-11; however, one can restrict the vertical beam width to reduce the reflected amplitude.

C. ROTATIONAL SCAN ANGLE MEASUREMENTS

A rotational scan experiment, similar to that conducted in the anechoic chamber, verified the detection capabilities of both the bare sonar pair, Sonar 0, and the coned sonar pair, Sonar 3, using MML 11 with the SPARC4 microprocessor. The rotational scan took place at approximately 73.5, 123.5, 173.5 and 223.5 centimeters from a continuous wall on the 5th deck of Spanagel Hall at the Naval Postgraduate School. Fortunately, multipath interference was not a problem at these distances. Instead of recording raw range data as in the previous experiment, the experiment recorded the global x-y coordinates of each data point. If no sonar return occurred in the allotted time period, the raw range was set to infinity, defined for Yamabico-11 as 1.0×10^6; if the range was infinity, the global coordinate did not print to the data file. Figures 30 and 31 plot these global data points.

At longer distances, the global coordinates of the return are more spread out due to the time required for sound to travel to and from the wall. Although the bare sonar pair produced smooth data, the coned sonar pair gave

49

inconsistent returns due to the narrowness of the beam and the presence of side lobes. The absence of returns occurred where the beam pattern of the coned sonar pair had a node. In both cases, at large angles, the sonars measured a shorter distance because the side of the beam generated the return vice the center of the beam. Since the global coordinate calculation assumed that the return occurred on beam centerline, the plots in Figures 30 and 31 curve at the edges.

Before this work began, others had observed that Yamabico-11 could not receive returns unless almost perpendicular to an object. At the time of these observations, Yamabico-11 had the sonar configuration of Figure 1 and used a 5 Volt supply voltage. Based on the beam patterns produced in the anechoic chamber, the original hypothesis made claimed that the cones caused this limitation. However, Figures 30 and 31 show virtually no difference between the measured angular response of the coned and bare sonar pairs using a supply voltage of 12 Volts. Previously, the amplitude of the returns generated by the side lobes using the 5 Volt supply were insufficient to trigger the Schmitt Trigger. With the increase in supply voltage, the side lobes are able to fire the Schmitt Trigger, giving a similar angular response as the bare sonar pair, disproving the original hypothesis.

Rotational Scan of Wall by Bare Sonar Pair
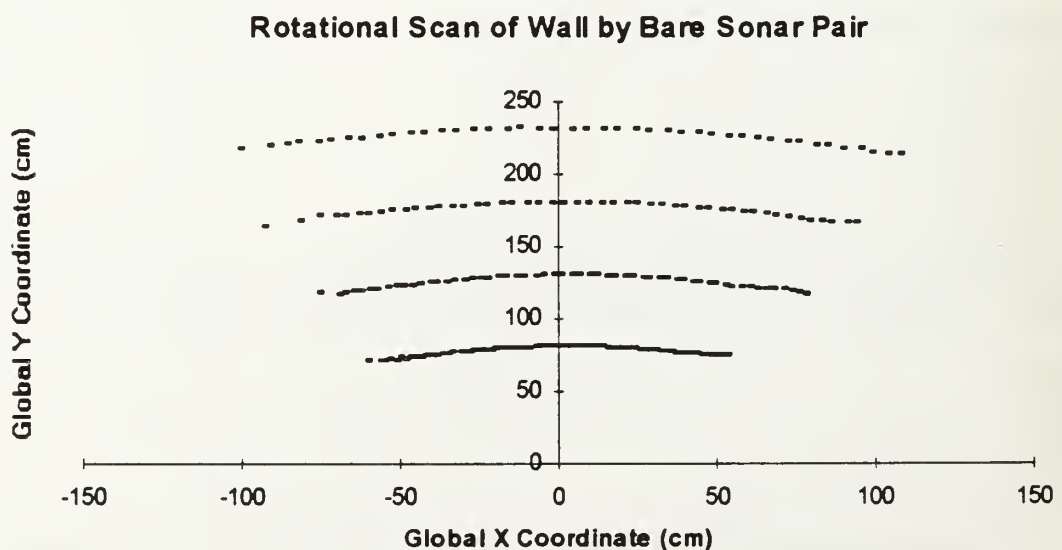
Figure 30
Rotational Scan of Wall by Bare Sonar Pair on Yamabico-11

Rotational Scan of Wall by Coned Sonar Pair
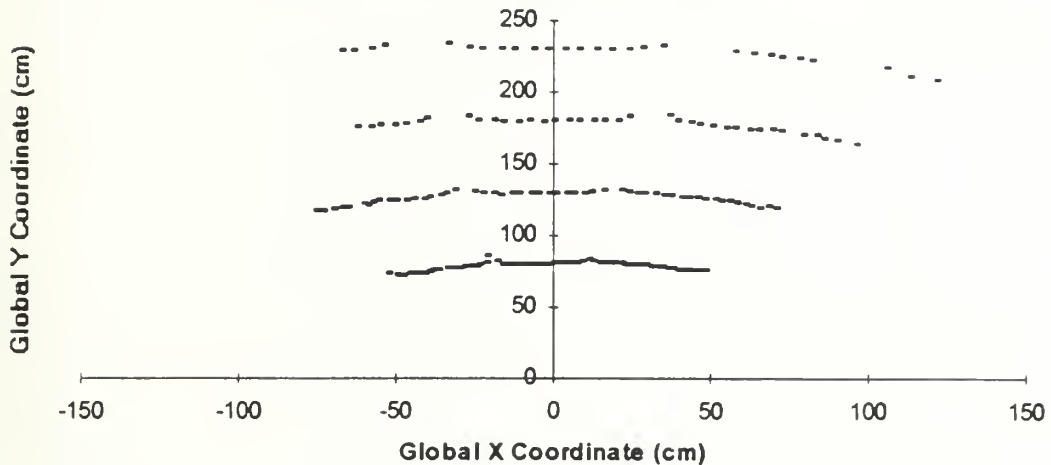
Figure 31
Rotational Scan of Wall by Coned Sonar Pair on Yamabico-11

D. OBSTACLE AVOIDANCE

The Yamabico-11 research group had demonstrated successfully various methods for obstacle avoidance using only ultrasonic sensor information. In the past, if the user programmed Yamabico-11 to move forward until it detected an object, there were two basic motions the user could use to avoid the object. The user, knowing both the location and size of the object, could calculate the obstacle avoidance path and program Yamabico-11 to maneuver around the obstacle using the pre-determined avoidance path. The user had to calculate the pre-determined avoidance path for each object encountered. Alternatively, the user could program Yamabico-11 to turn right or left and perform a wall-hugging motion to maneuver around an object.

Although both of these methods have been demonstrated, they each have drawbacks. In the first case, the user had to determine the proper avoidance path for each object. If a new object was added, or the starting configuration changed, the avoidance paths had to be recalculated and reprogrammed. In the second case, a simple wall-hugging heuristic in which Yamabico-11 turned right

when it detected an object could cause Yamabico-11 to take the long path around an object as Figure 32 demonstrates. Combining these approaches will give Yamabico-11 the ability to determine its own intelligent obstacle avoidance path, bringing it one step closer to being truly autonomous.
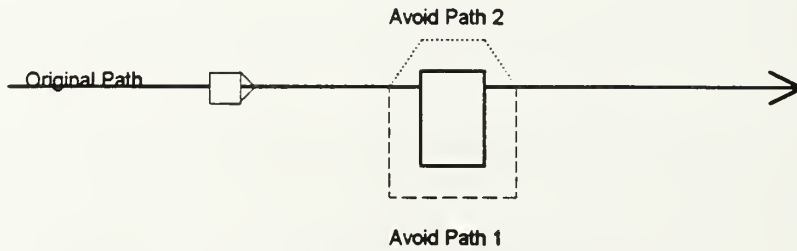
Figure 32. Two Obstacle Avoidance Paths

In order to determine its obstacle avoidance path, Yamabico-11 needs information about the size and location of the obstacle. With this information, it can decided the best path for avoiding the object intelligently given its *a priori* knowledge of the world and its desired path. Figure 33 shows a generic
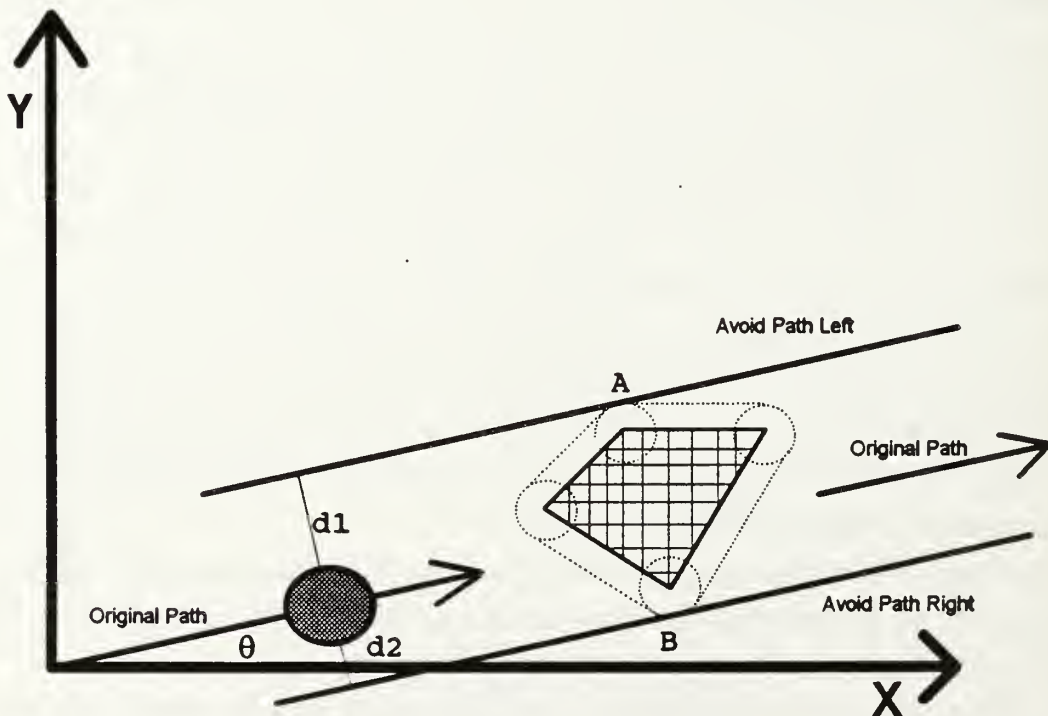
Figure 33. Geometry of the Obstacle Avoidance Problem

obstacle avoidance situation for a convex polygonal object. This method uses an avoidance path that is parallel to the original path to minimize robot movements. If the object is known, then the coordinates of the vertices A and B are known and intelligent obstacle avoidance is implemented easily. As can be seen in Figure 33, the obstacle has been grown by a safety factor. The signed distances d1 and d2 are calculated by the formula

$$d = -(x_i - x_c)\sin\theta + (y_i - y_c)\cos\theta + r \qquad \text{(Eq. 6-5)}$$

where the subscript i represents Point A or B, the subscript c stands for the robot's center position, and r is the safety margin. A positive d means the point is to the left of the robot; a negative signed distance places the point to the robot's right.

To calculate the avoidance path, the magnitudes of d1 and d2 are compared. Ideally, the robot should maneuver to the short side to avoid the object. Knowing the appropriate signed distance d, the avoidance path configuration is defined by

$$((x,y),\theta,\kappa)$$

where

$$x = x_c - d\sin\theta$$
$$y = y_c + d\cos\theta \qquad \text{(Eq. 6-6)}$$

where the orientation, θ, is that of the original path and curvature, κ, is zero, defining a straight line.

If the obstacle detected does not correspond to any known object, then other means are needed to get the required information. Although Yamabico-11's current sonar system can easily detect the presence of an object, it is not suitable for determining the object size quickly. Using only its sonar system, Yamabico-11 would have to physically move, testing for the object, in order to determine the outer boundaries of the object. This movement is time consuming and inefficient. A better method for localizing an object requires the use of a sensor which can detect the width of an object.

One solution is to use a visual system which has the ability to determine the projection of an object onto a vertical plane. Given the range and the global x - y coordinates of an object, a visual system could determine the widths, d1

53

and *d2*, in Figure 33, assigning a negative width to *d2*. Combining this information with knowledge of the robot's environment map and desired path allows calculation of a safe and appropriate path to avoid the obstacle using Equation 6-6.

Determining when the robot is past the object and can transition safely back to the original path is difficult. By using a side facing sonar, the robot can ensonify the object, checking for a distance greater than the magnitude of the signed distance *d*. A problem arises if the sonar is enabled *before* the robot has maneuvered abreast of the object. The time required for the robot to maneuver to the avoidance path is a factor of both the magnitude of the signed distance and the robot's speed.

The robot will be abreast the initial detection point of the obstacle when

$$x_c = x_a + d\cos\theta$$
$$y_c = y_a + d\sin\theta$$

(Eq. 6-7)

where x_c and y_c are the global coordinates of the robot at any instant in time, x_a and y_a are the global coordinates of the avoidance path defining point, *d* is the detection distance, and θ is the orientation of the original path. Once the robot has reached this point, it can enable the side-facing sonar safely.

Figures 34 and 34 show the path followed by Yamabico-11 in maneuvering around an object. In Figure 34, the *(x,y)* object boundaries given to Yamabico-11 were (150.0,105.0) and (150.0, 25.0). The initial path started at the origin at an orientation of 30° and with zero curvature. Yamabico-11 correctly calculated an avoidance path to the left of the object and returned to the original path once past the object. In Figure 35, the coordinates were (150.0,175.0) and (150.0,55.0) and the initial path was the same. This time, Yamabico-11 decided to maneuver to the right to avoid the object.

Obstacle Avoidance to the Left
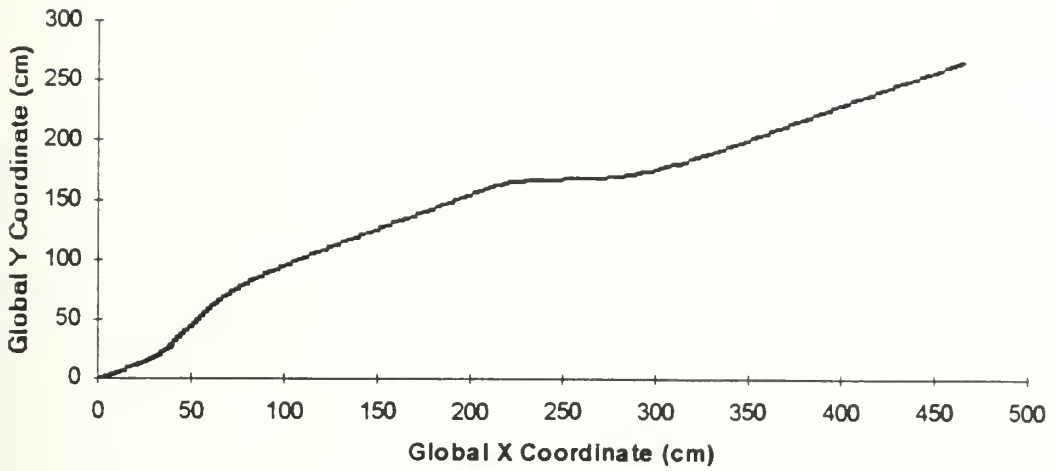
Figure 34
Dynamic Obstacle Avoidance to the Left

Obstacle Avoidance to the Right
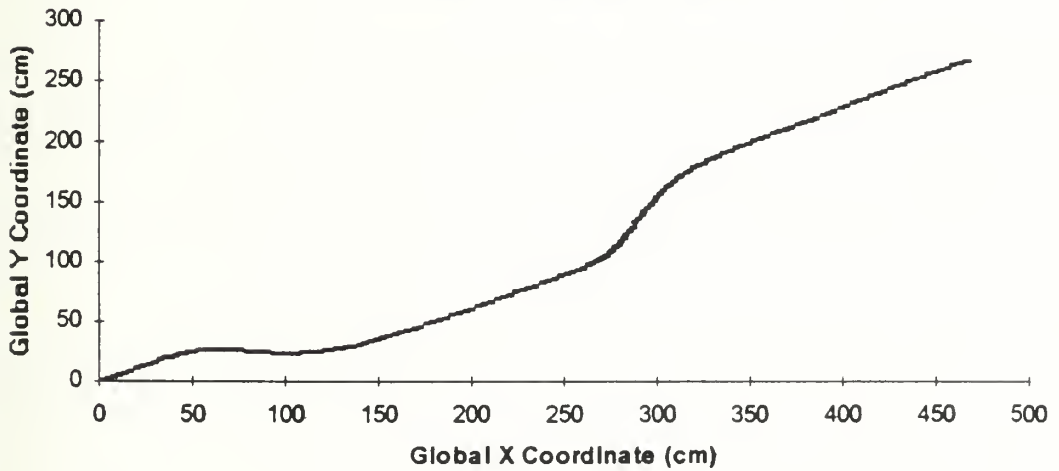
Figure 35
Dynamic Obstacle Avoidance to the Right

55

VII. NEW SONAR DESIGN

A. JUSTIFICATION

The results of the experiments in the anechoic chamber and on Yamabico-11 show the necessity of re-configuring the sonar system on Yamabico-11 to provide consistent sonar returns and to provide full 360° coverage throughout its operating range. Simply removing the cones from the sonar pairs will allow consistent sonar returns. Repositioning the twelve sonar pairs evenly around the periphery of the robot at 30° increments will provide the most comprehensive coverage. Overlapping the sonar coverage provides an added benefit in that returns from neighboring sonar pairs can be compared to give a gross estimate of obstacle orientation.

B. DESIGN

1. Hardware System Modifications

The sonar suite was modified as follows:

       a. Removed cones from all sonar pairs.

       b. Moved Sonars 0, 2, 5 and 7 to centerline front, back, left and right, respectively.

       c. Moved Sonars 3, 1, 4, 6 to the right of Sonars 0, 2, 5 and 7, respectively, at a 30° angle clockwise from the centerline sonar pair on each side.

       d. Moved Sonars 11, 10, 8 and 9 to the left of Sonars 0, 2, 5 and 7, respectively, at a 30° angle counter-clockwise from the centerline sonar pair on each side.

Figure 36 shows the new locations of the sonars. This new system configuration gives Yamabico-11 full 360° sonar coverage with consistent sonar returns.
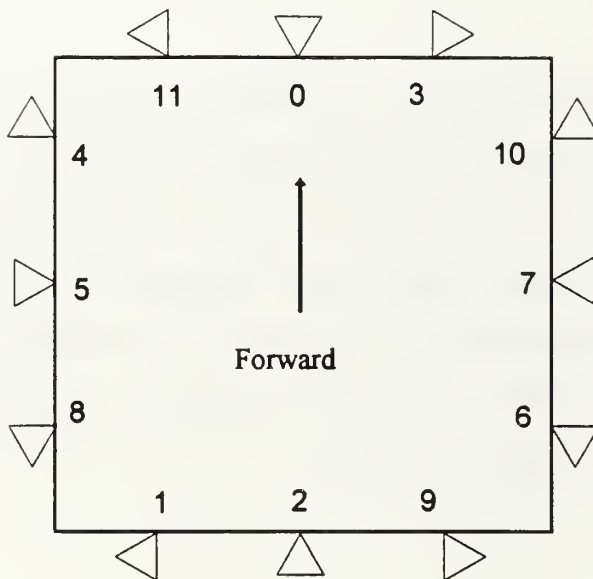
Figure 36
New Sonar Pair Locations

2. Software System Modifications

The file "sonar.c" in MML 11, contains the sonar table with the sonar location information. This table reflects the new sonar positions in Figure 36. New mnemonics, based on relative position of each sonar, describe the sonars in MML 11. Table 3 lists each sonar pair, its mnemonic and its group.

A new function called "ScanSonar (int, double)" created in MML 11 allows the user to scan for obstacles in one of four directions: forward, backward, left or right. This function automatically pings the appropriate sonar pairs in the direction specified, giving the user the global x-y coordinates of the first obstacle detected within the user-specified range. Another function called "avoidPathVertex (double, POINT)" gives Yamabico-11 the ability to determine dynamically an avoidance path based on the vertices of a convex polygons as

described in Chapter VI while "avoidPathWidth(double, POINT)" gives a dynamic avoidance path based on the width of an object relative to the robot's orientation. The vertices and/or widths are provided by the dummy functions "getBoundaries(POINT)" and "getWidth(POINT,int)"; although these functions return values, the values must be inputted manually by the user. However, the structure exists for future implementation of these functions.

Additionally, modifications to the variable naming convention in the sonar table located "sonar.c" make it more readable. The sonar positional information is now named "sonartable[n].SonarPosit.X", "sonartable[n].SonarPosit.Y" and "sonartable[n].Theta" where n stands for the sonar number. Modifications to the files "sonar.h" and "sonarmath.c" ensured the consistency of these changes.

Mnemonic	Sonar	Group
S000	0	0
S030	3	1
S060	10	2
S090	7	0
S120	6	1
S150	9	2
S180	2	0
S210	1	1
S240	8	2
S270	5	0
S300	4	1
S330	11	2

Table 3
New Sonar Mnemonics

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. BEAM WIDTH

This work investigated the sonar hardware on Yamabico-11, including its characteristics and limitations, and will serve as a major reference for further improvements to the sonar system. Increasing supply voltage to the sonar driver boards from 5 volts to 12 volts caused the effective beam width to increase because the side lobes now provided usable returns. However, the data was inconsistent due to the nodes in the beam pattern. Removing the cones eliminated the side lobes, providing consistent range data. The effective beam width without the cones is about the same as it was with the cones. The wider beam width produced by the increased supply voltage has the negative effect of introducing multipath interference. The wider beam width improves the sonar system coverage, but signal cancellation reduces the range of the sonar system. The range can be increased by increasing the height of the sensors, but this also increases the minimum detection distance of objects near the floor. Using a height of about 36.5 centimeters for the sensors, consistent range data is achievable out to a range of over 2 meters. Over this range, the data return is intermittent due to the cancellation affects of multipath interference. An even re-distribution of the twelve sonar pairs around the periphery ensures that Yamabico-11 has 360° sonar coverage. With twelve evenly spaced sonars, full coverage can be achieved with a minimum full beam width of 30°. A smaller beam width will reduce the multipath interference effects. This smaller beam width can be achieved by placing the sensors in a properly shaped horn. Investigation of the proper shape for the horn is left for future work.

B. SIGNAL PROCESSING

The current signal processing, which uses a threshold to determine detection of a return signal, limits the ability of Yamabico-11 to calculate sonar ranges accurately . The current threshold, set at 1.4 Volts, may or may not be reached by the time the return signal hits its maximum at 0.5 milliseconds after the return signal first reaches the receiver. Therefore, thresholding causes a

61

dynamic error of up to ± 8.6 centimeters. To remove dynamic error, the signal processing must change. Since the occurrence of the return signal's peak is predictable and constant, it could be used to stop the clock counter, thereby ensuring that the clock counter stops at the same point for each pulse regardless of distance to, or material composition of, an object. New signal processing could use either analog or digital circuits to remove the dynamic error. A recommended solution could use a circuit which detects a change in the sign of the return signal's slope. Also, if this circuit maintains the voltage amplitude of the return signal's peak, Yamabico-11 could perform even more sophisticated signal processing, using signal strengths from different sonar pairs to localize an object. Further investigation of these and other signal processing techniques will greatly enhance the sonar system on Yamabico-11 and allow Yamabico-11 to move more precisely within its world.

Although much smaller, the system has static errors caused by hardware timing constraints. Once the dynamic error is removed, the clock-distance conversion algorithm can be modified to account for the static offsets. Ideally, this would allow the accuracy of the sonar ranges to be within one clock tick or ± 0.1029 centimeters.

APPENDIX A.
DATA FROM ANECHOIC CHAMBER EXPERIMENTS

This appendix contains the analog graphs recorded by the HP 7090A Plotter during the anechoic chamber experiments discussed. In each case, the rotator speed was 1 rpm. The direction of rotation varied for each experiment. Each graph has the x-axis divided into 10° increments. The peak voltage occurs at 0°. This series of experiments compares the beam widths determined by the x-axis. The y-axis scale varies among the graphs, but no comparison of voltage is made among the graphs. The voltage is only a factor in determining the half-amplitude point.

Graph A corresponds to Experiment 1 in Chapter V. This data was collected on April 24, 1994. The bare Nicera transmitter was rotated while the bare receiver was held steady at a distance of about 199.3 centimeters.

Graph B corresponds to Experiment 2 in Chapter V. This data was collected on April 24, 1994. The transmitter with small cone was rotated while a bare receiver was held steady at a distance of about 198.0 centimeters.

Graph C corresponds to Experiment 3 in Chapter V. This data was collected on May 20, 1994. The receiver with large cone was rotated while a bare transmitter was held steady at a distance of about 200.0 centimeters.

Graphs D, E and F support to Experiment 4 in Chapter V. The data was collected by rotating a coned sonar pair with a wall located at a distance of about 76 centimeters. For Graph D, produced on April 7, 1994, the coned sonar pair was mounted horizontally with the transmitter t the left of the receiver. Graph E and F, produced on April 13, 1994, mounted the coned sonar pair vertically. In Graph E, the receiver was above the transmitter and in Graph F, the receiver and transmitter were reversed. The beam pattern produced in all three graphs was the same.

Graphs G, H, and I support Experiment 5 in Chapter V and were produced on April 13, 1994. In this series, a bare sonar pair was rotated about 80 centimeters from the wall. In Graph G, the bare sonar pair was mounted vertically with the receiver above the transmitter. In Graphs H and I, the bare sonar pair was mounted horizontally with the transmitter to the left of the receiver. In Graph H, the bare sonar pair was rotated counter-clockwise and

rotated clockwise in Graph I. In all three cases, a similar beam pattern was produced.
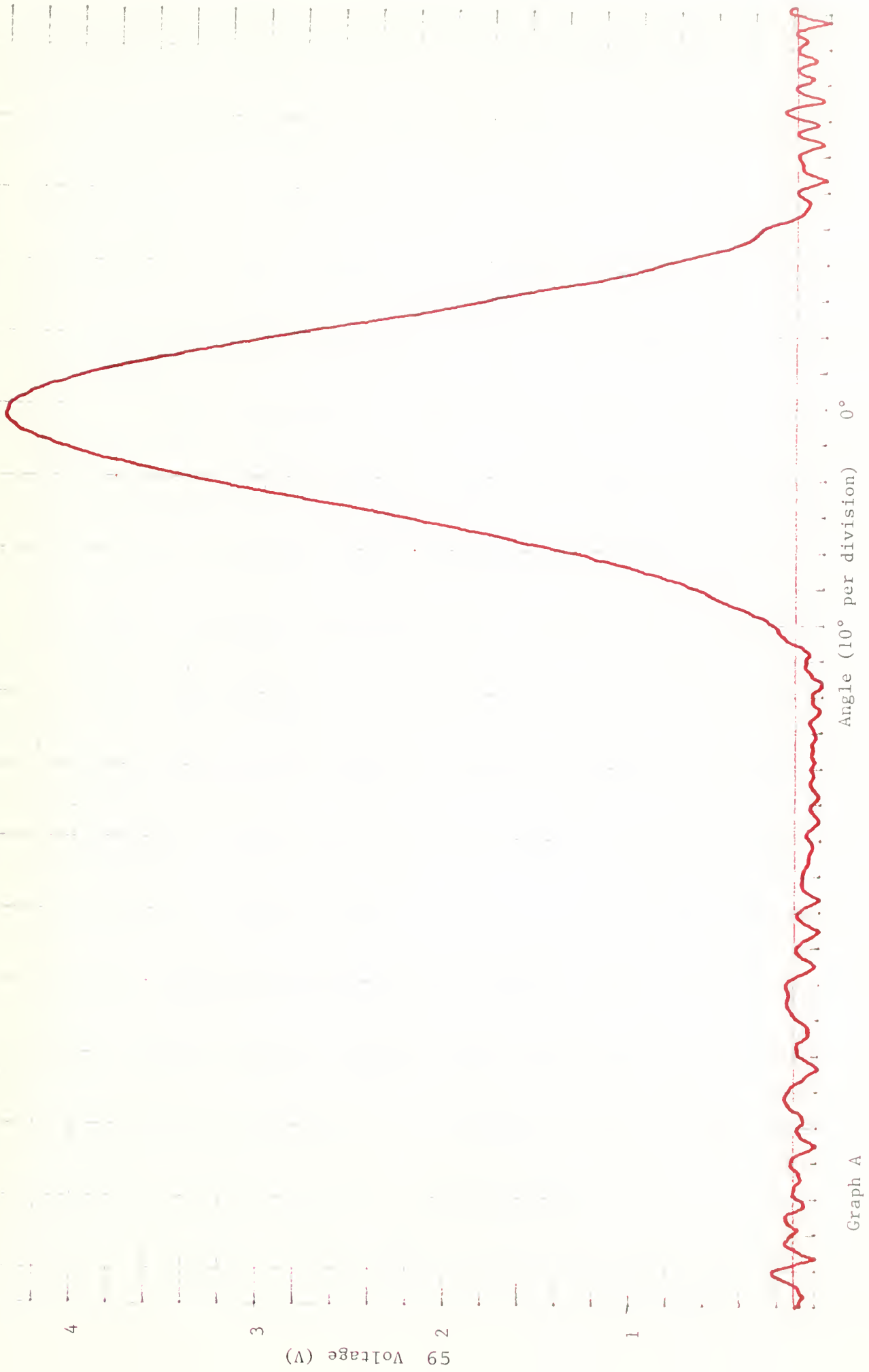
Graph A
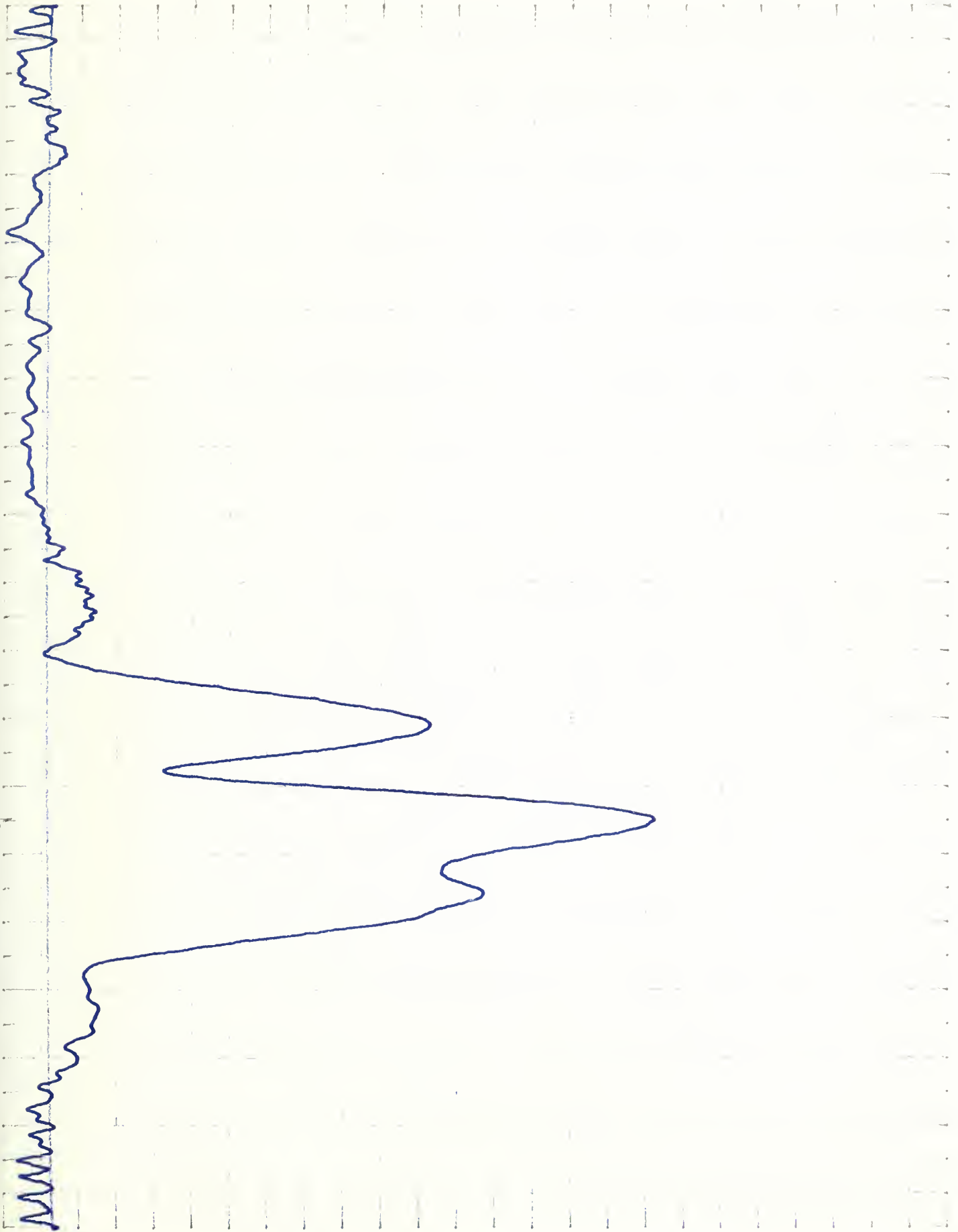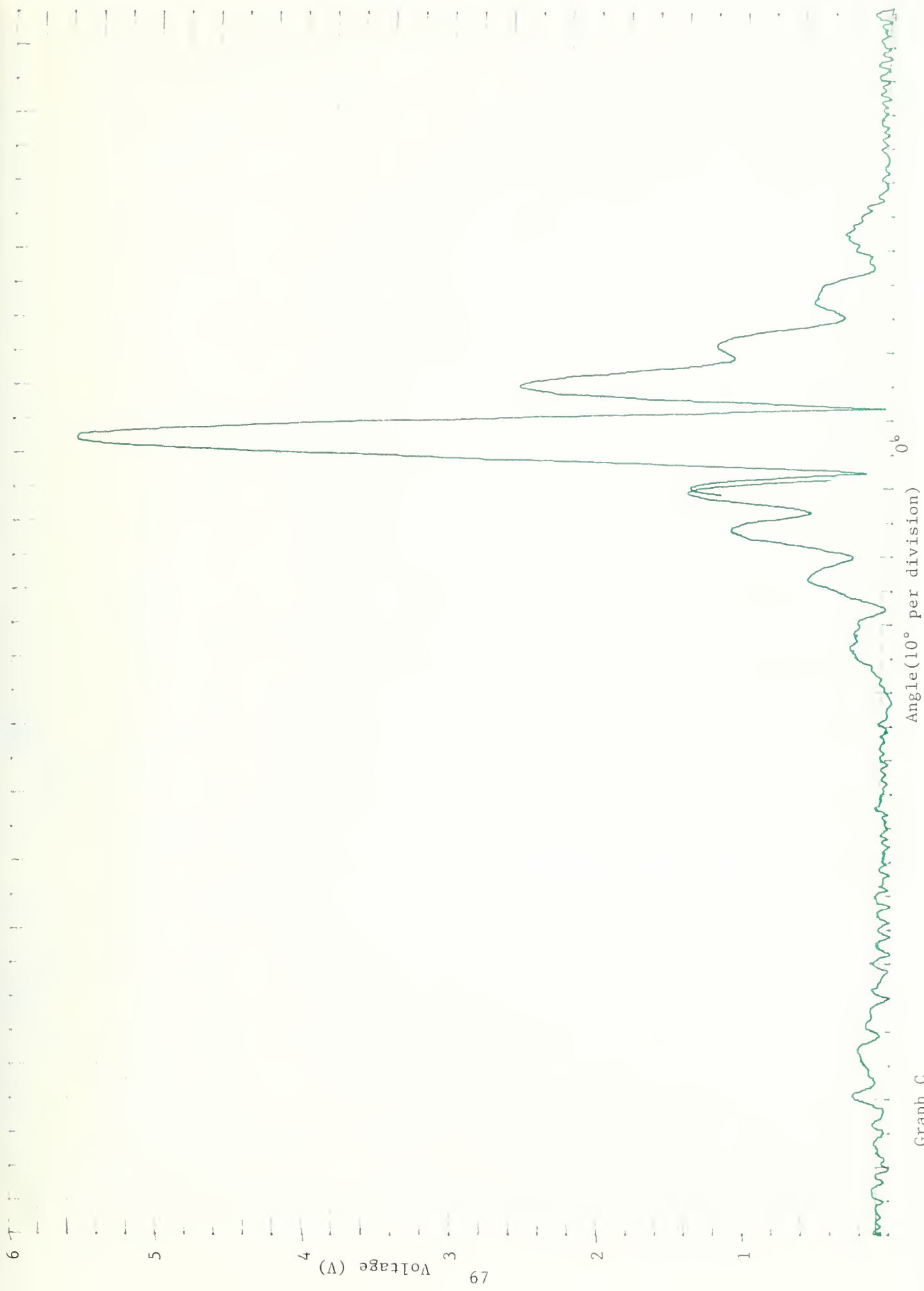
Angle (10° per division)   0°

Voltage (V)

4   3   2   1

Voltage (V)

66

Graph C

Angle(10° per division)

Voltage (V)

67

Voltage (V)

68

Graph E

Angle (10° per division)

Voltage (V)

69

1   2   3   4   5   6

Graph 1

Graph G

Angle (10° per division)

Voltage (V)

71

Voltage (V)

72

Graph I

Angle (10° per division)

0°

Voltage (V)

73
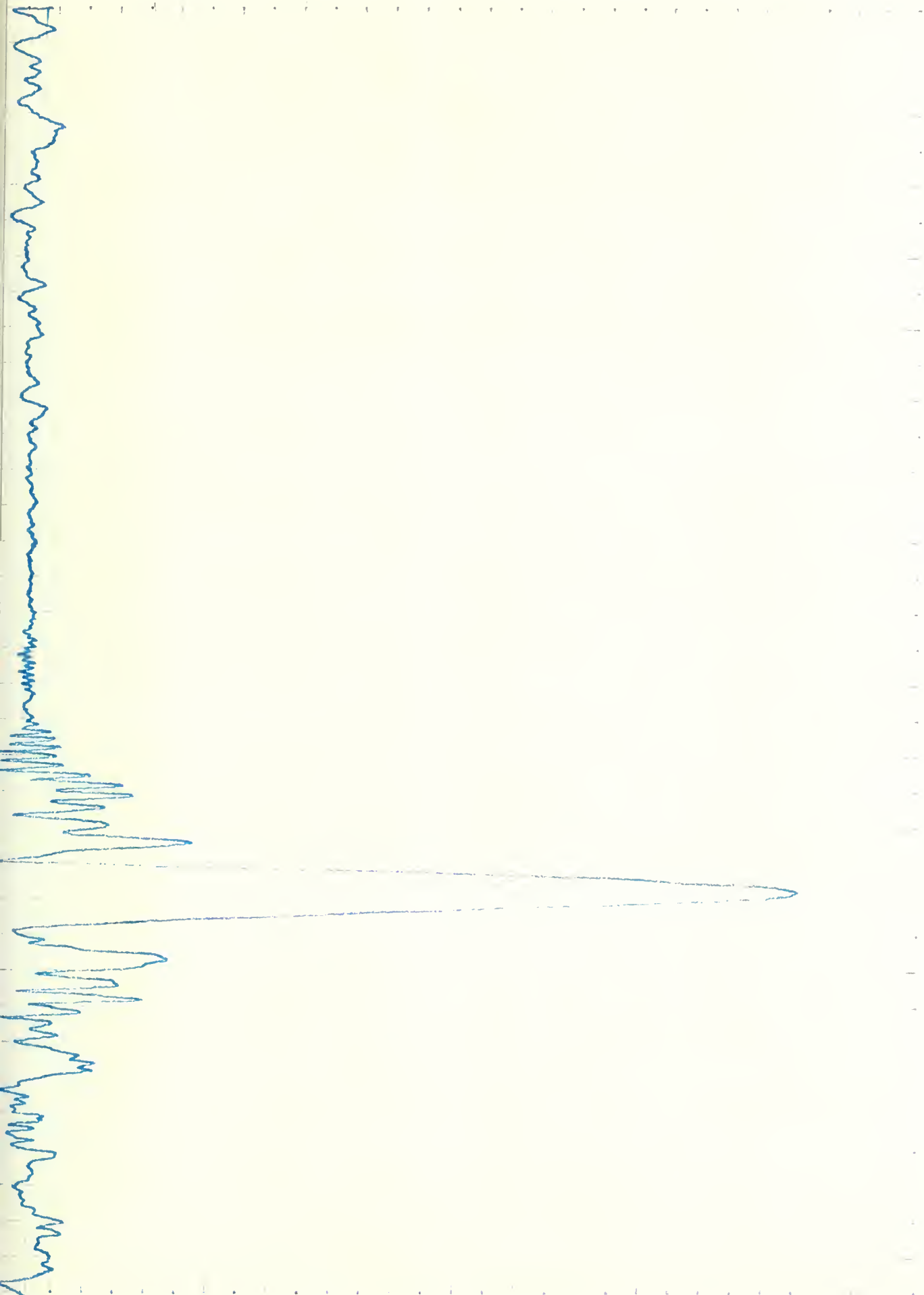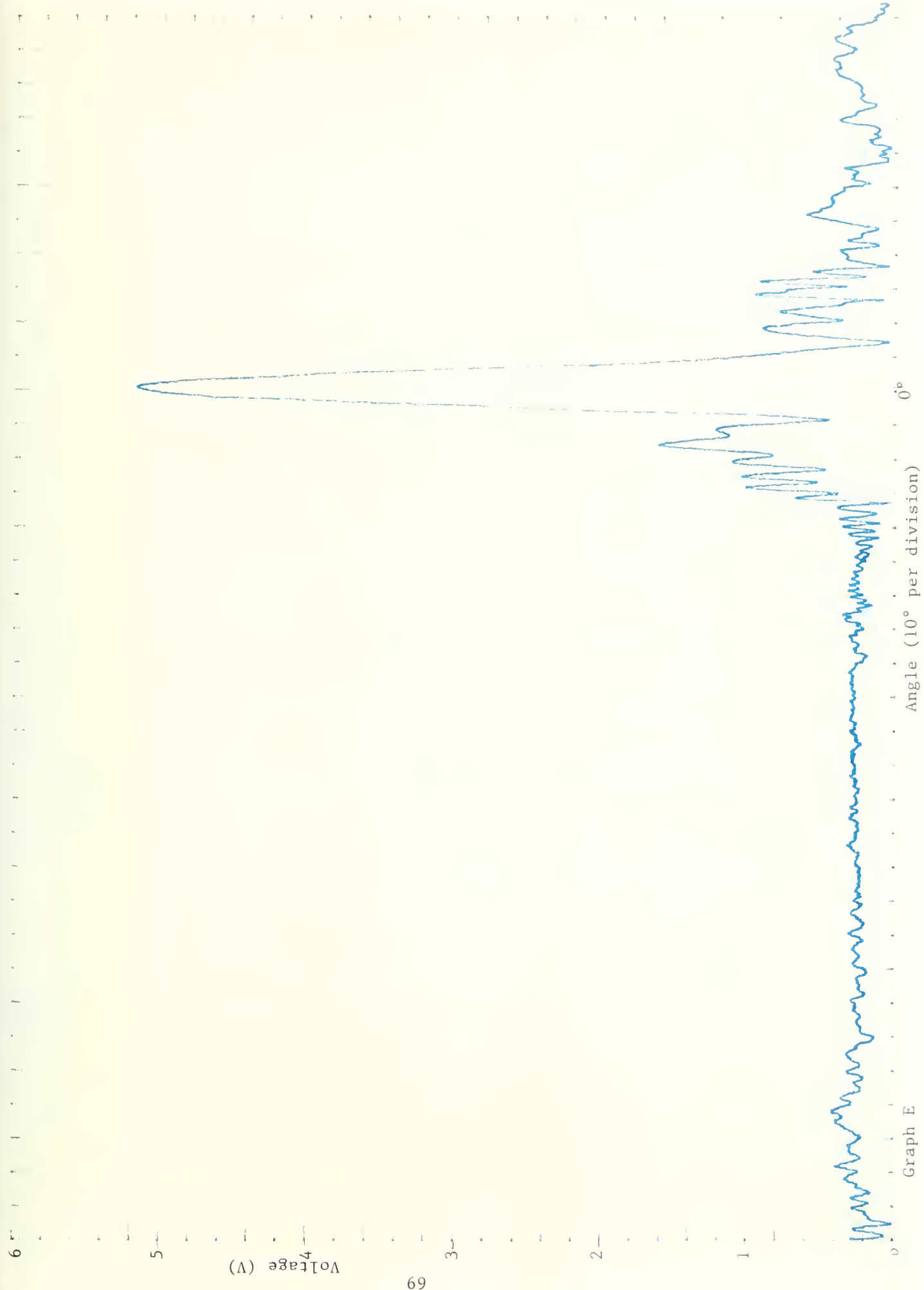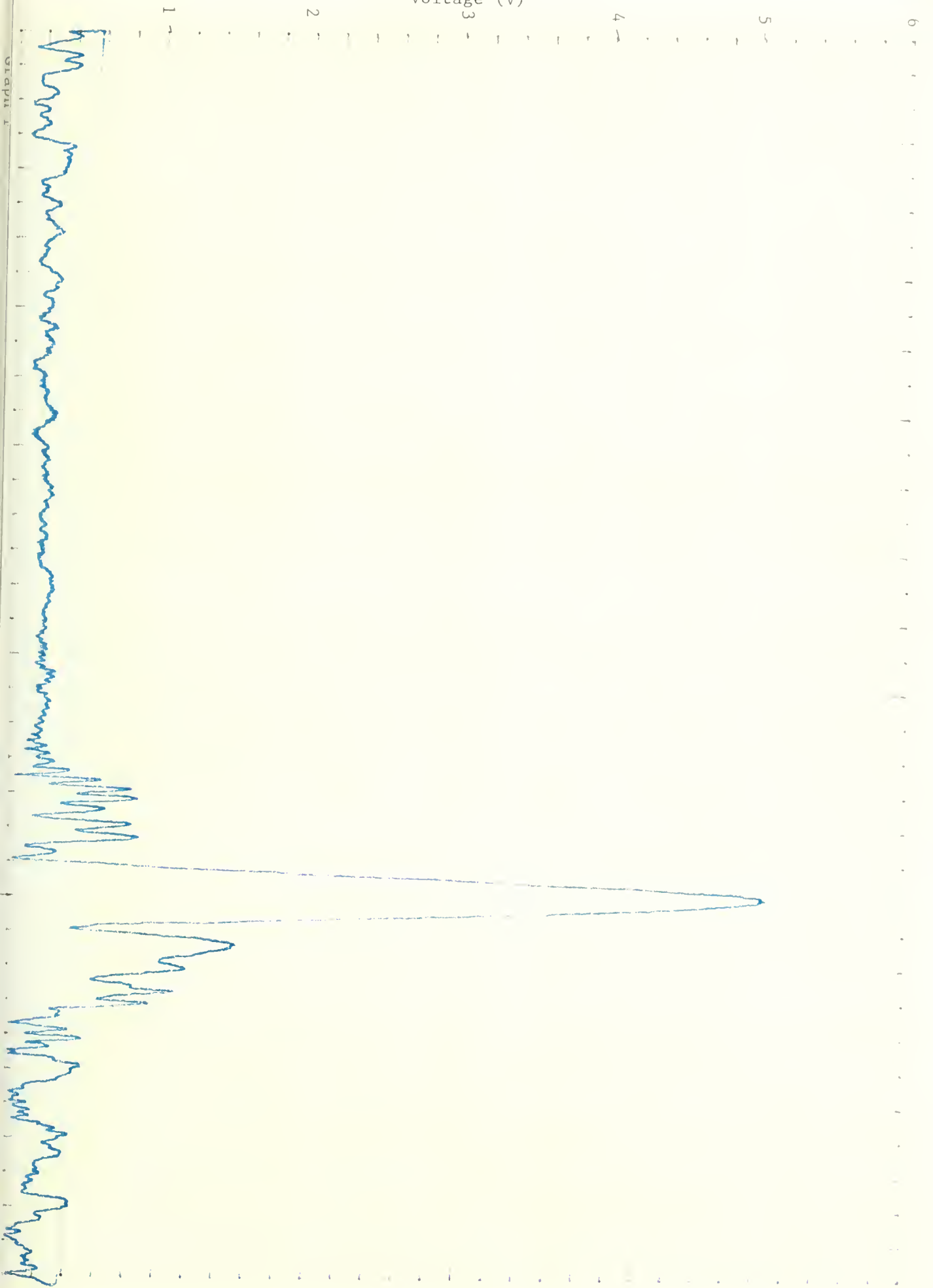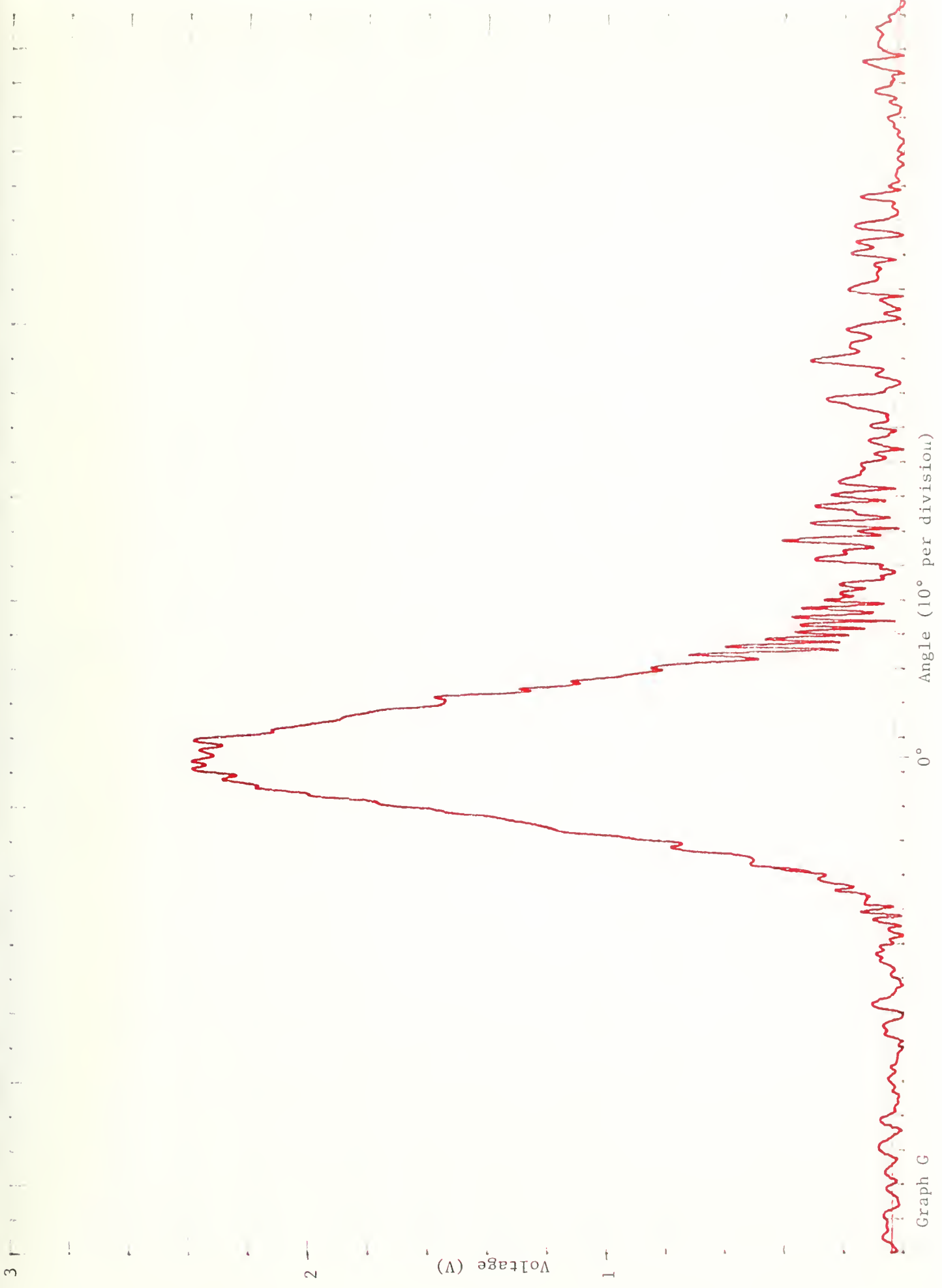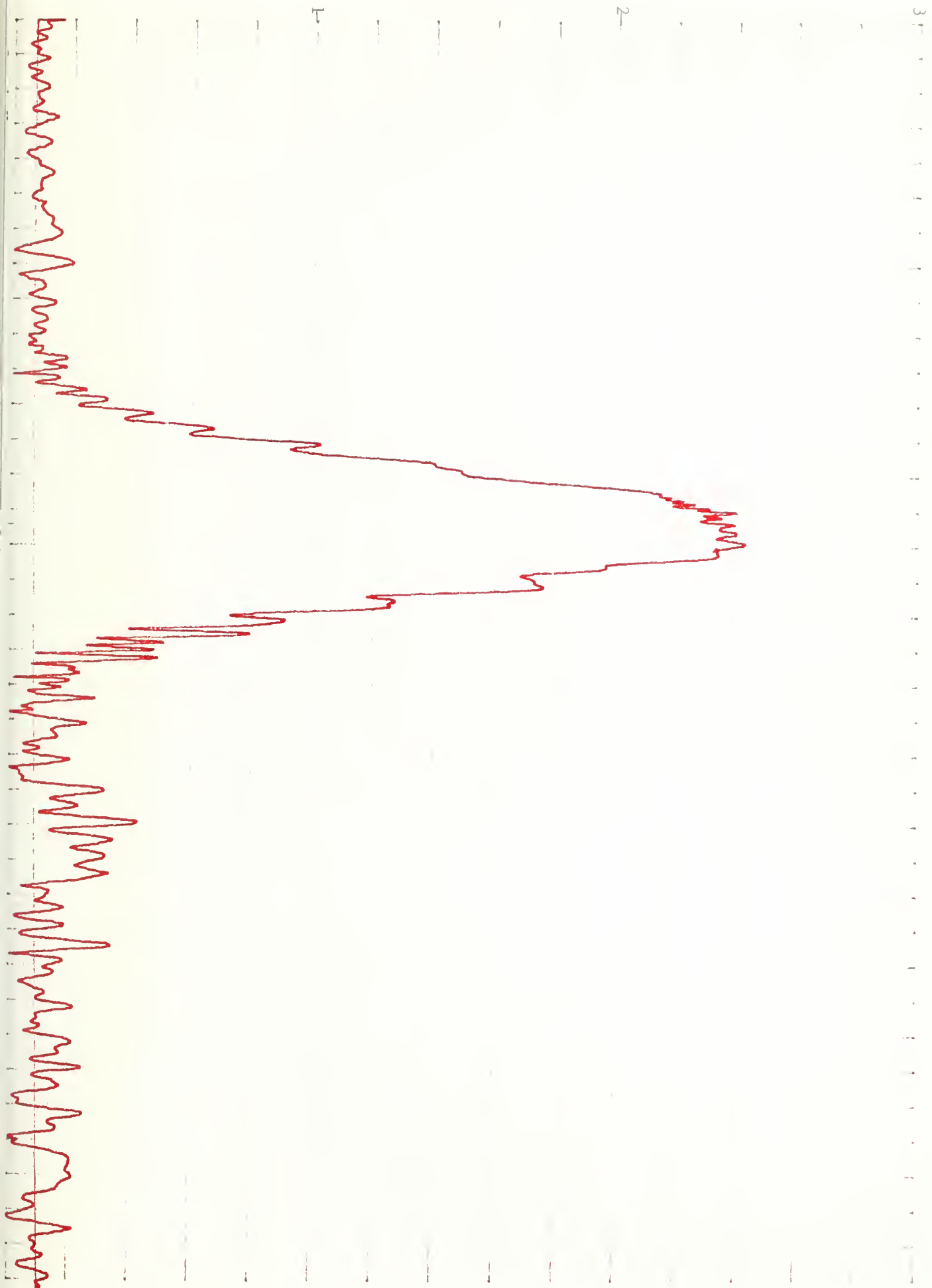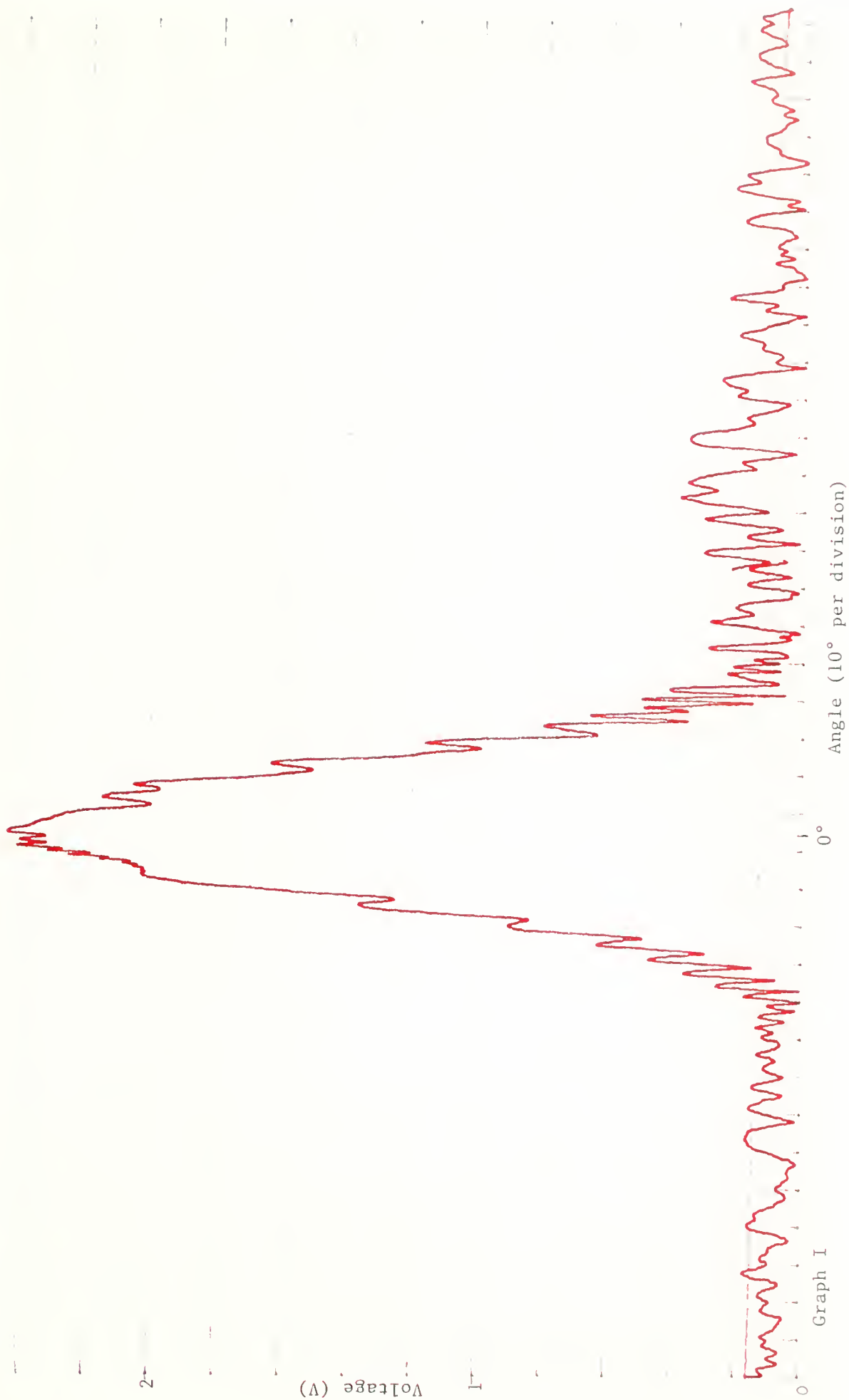
APPENDIX B.   USER PROGRAMS

This appendix contains the user programs written to perform testing on Yamabico-11. Each file contains an explanation in the heading and indicates the MML version. User files written in MML 10 and MML 11 differ greatly. Comments within each user file explain the logic.

```
/*********************************************************
*
*   User File:    DistTest
*   Description: Tests distance accuracy of sonar
*   MML Version: MML 10
*   Author:       Jane Lochner
*   Date:         15 JUL 94
*   Notes:        Change #define statement to reflect
*                 sonar number or mnemonic of sonar
*                 under investigation.  Once program
*                 is run, be sure to rename the data
*                 file "RAW" on the host before
*                 running the program again.
*********************************************************/


#include "mml.h"

#define SONAR 3
#define FILETYPE 0
#define FILENUMBER 0

User()
{
  long int i;
  void initialize();
  void cleanup();

  initialize();

  motor_on=OFF;

  do
    {i++;
     sonar(SONAR);}      /*  Gives 21 readings */
    while (i<50000);

  cleanup();

}


void initialize()        /* sets up data logging    */
  {
    enable_sonar(SONAR);
    set_log_interval(SONAR,1);
    enable_data_logging(SONAR,FILETYPE,FILENUMBER);
  }

void cleanup()           /* transfers data to host  */
  {
    disable_sonar(SONAR);
    disable_data_logging(SONAR,FILETYPE);
    xfer_raw_to_host(FILENUMBER,"RAW");
  }
```

```
/*****************************************************
 *
 *  User File:    ContSonar
 *  Description:  Continuously pings given sonar
 *  MML Version:  MML 11
 *  Author:       Jane Lochner
 *  Date:         6 OCT 94
 *  Notes:        Change #define statement to reflect
 *                sonar number or mnemonic of sonar
 *                under investigation. Program must
 *                be stopped using the manual interrupt
 *                switch on Yamabico-11.
 *****************************************************/

#include "user.h"


#define SONARUSED 0


void
user()
{
  EnableSonar(SONARUSED);

  do
    {
     Sonar(SONARUSED);
     }
   while (TRUE);
   DisableSonar(SONARUSED);
}
```

```
/*********************************************************
 *                                        .
 *  User File:    Scan
 *  Description: Tests sonar scanning function
 *  MML Version: MML 11
 *  Author:      Jane Lochner
 *  Date:        15 OCT 94
 *  Notes:       Prints out to the screen the
 *               Global X-Y Coordinates of object
 *               detected.
 *********************************************************/


#include "user.h"


void
user()
{
    POINT obstacle;

    obstacle = ScanSonar(FORWARD,150.0);
    printf("X coord is:  %f\nY coord is:  %f\n\n",obstacle.X,obstacle.Y);

}
```

```
/********************************************************
 *
 *   User File:    RotScanWall
 *   Description:  Records the Global X-Y Coordinates
 *                 of objects detected.
 *   MML Version:  MML 11
 *   Author:       Jane Lochner
 *   Date:         7 OCT 94
 *   Notes:        Change #define statement to reflect
 *                 sonar number or mnemonic of sonar
 *                 under investigation. Used to determine
 *                 the angular response of a given sonar
 *                 by facing the sonar down the hallway,
 *                 then rotating 180 degrees.  Infinite
 *                 returns can be deleted from the data
 *                 file and results plotted.
 ********************************************************/

#include "user.h"


#define SONARUSED 3
#define USERBUFSIZE 0
#define FREQ 1
#define MODE SONAR_GLOBAL

void
user()
{
  CONFIGURATION Start,Current;
  POINT obstacle;


  EnableSonar(SONARUSED);
  SonarLog(FREQ,USERBUFSIZE,SONARUSED,MODE);


  Start = defineConfig(0.0,0.0,0.0,0.0);
  setRobotConfigImm(Start);
  setRotVelImm(5.0);

  Current = getRobotConfig();
  Rotate(PI);

  waitSec(10);

}
```

```c
/**********************************************************
 *
 *  User File:    SonarTest
 *  Description:  Gives distance from sonar face to
 *                object in centimeters
 *  MML Version:  MML 11
 *  Author:       Jane Lochner
 *  Date:         15 SEP 94
 *  Notes:        Change #define statement to reflect
 *                sonar number or mnemonic of sonar
 *                under investigation.
 **********************************************************/

#include "user.h"


#define SONARUSED 3
#define USERBUFSIZE 0
#define FREQ 5
#define MODE SONAR_RAW

void
user()
{   .

   long int i;
   EnableSonar(SONARUSED);
   SonarLog(FREQ,USERBUFSIZE,SONARUSED,MODE);


   do
     {
      LogSonarData(SONARUSED);
       i++;}                              /*  Gives 7 distance readings */
     while (i<50);

   DisableSonar(SONARUSED);

}
```

```
/****************************************************
 *
 *  User File:    demo
 *  Description: Robot moves until 1 meter from
 *               an object then executes a 135 degree turn.
 *               Robot will repeat this maneuver until
 *               the manual interrupt button is pushed.
 *  MML Version: MML 11
 *  Author:      Jane Lochner
 *  Date:        29 SEP 94
 *  Notes:       Change #define statement to reflect
 *               sonar number or mnemonic of sonar
 *               under investigation.
 ****************************************************/


#include "user.h"


#define SONARUSED 0


void
user()
{

  int GOING = 1;
  double hit;
  CONFIGURATION  Start, JustGo, CurrentPosit, Jump, NewPosit;

  Start = defineConfig(0.0, 0.0, 0.0, 0.0);
  JustGo = defineConfig(0.0, 0.1, 0.0, 0.0);
  Jump = defineConfig(0.0, 45.0,-1.5*HPI, 0.0);


  EnableSonar(SONARUSED);

  setLinVelImm(15.0);

  setRobotConfigImm(Start);

  hit = 9999.9;

  line(JustGo);

  do{
        while(hit >=100.0 || hit <= 1.0)
            {
                hit = Sonar(SONARUSED);
                printf("\n Range is:  %f ",hit);
            }

        CurrentPosit = getRobotConfig();
                            NewPosit = compose(&CurrentPosit,&Jump);

        setRobotConfigImm(NewPosit);

        hit = 9999.9;

        printf("\n\nNew Sonar Range is %f\n",hit);
        waitSec(2);

    }while(GOING);
}
```

```
/***********************************************************
 *
 *   User File:      avoidPathVertex
 *   Description: Tests ability of Yamabico to determine
 *                obstacle avoidance path autonomously.
 *   MML Version: MML 11
 *   Author:         Jane Lochner
 *   Date:           26 NOV 94
 *   Notes:          Change #define statements to reflect
 *                   sonar number or mnemonic of sonar
 *                   under investigation, safety factor, and
 *                   obstacle notification distance
 ***********************************************************/
#include "user.h"

#define SONARUSED 0
#define USERBUFSIZE 0
#define FREQ 10
#define MODE SONAR_RAW
#define SAFETY 20.0
#define DISTANCE 150.0
#define DIRECTION FORWARD

void
user()
{
  double width,hit,distance;
  CONFIGURATION Start,JustGo,Avoid,vehicle;
  POINT obstacle;
  int sonar;

  Start = defineConfig(0.0,0.1,PI/6,0.0);
  JustGo = defineConfig(0.0,0.0,PI/6,0.0);

  MotionLog(NULL,25,0);

  EnableSonar(SONARUSED);

  setRobotConfig(Start);
  setLinVelImm(15.0);

  line(JustGo);

  obstacle = ScanSonar(DIRECTION,DISTANCE);

  vehicle = getRobotConfig();

  Avoid = avoidPathVertex(SAFETY,obstacle);

  width = -(Avoid.Posit.X-vehicle.Posit.X)*sin(vehicle.Theta)+
           (Avoid.Posit.Y-vehicle.Posit.Y)*cos(vehicle.Theta);


  if (width < 0)
             {
      width = width - SAFETY;
      sonar = S270;
      distance = -width;
    }
  else
    {
      width = width + SAFETY;
```

82

```
          sonar = S090;
          distance = width;
                     }
    line(Avoid);

    while
((vehicle.Posit.X <(Avoid.Posit.X+DISTANCE*cos(JustGo.Theta)))
        ||
(vehicle.Posit.Y < (Avoid.Posit.Y+DISTANCE*sin(JustGo.Theta))))

                       {
           vehicle = getRobotConfig();
           waitMS(500);
           }

    EnableSonar(sonar);
    hit = Sonar(sonar);
    waitMS(30);
    while(hit <= distance)
          {
          hit = Sonar(sonar);
printf("Distance = %f\n",hit);
          waitMS(30);
          }

    line(JustGo);
    waitSec(20);
    stopImm();
}
```

APPENDIX C.   MML11 LIBRARY FILES

This appendix contains the MML11 files which contain changes resulting from this work. The heading of each new function explains the inputs and outputs of the functions and describes the function's use.

```
/*****************************************************
 *  File:       sonar.h
 *  Comments:  12-07-94  Updated for new sonar naming
 *                       convention by Jane Lochner.
 *****************************************************/
#ifndef __SONAR_H
#define __SONAR_H

#include "constants.h"
#include "definitions.h"

#define    NUM_SONARS       16

/* Sonar locations */
#define S000            0
#define S030            3
#define S330           11
#define S090            7
#define S060           10
#define S120            6
#define S180            2
#define S150            9
#define S210            1
#define S270            5
#define S240            8
#define S300            4

/* Types of sonar logging */
#define    SONAR_NONE         0x00
#define    SONAR_RAW          0x01
#define    SONAR_GLOBAL       0x02
#define    SONAR_SEGMENT      0x04
#define    SONAR_ALL          0x07


#define SONAR_CTL          0xfc0083f9


typedef struct  {
    int     fitting,            /*flag to indicate linear fitting request      */
            globalCoord,        /*flag to indicate coordinate conversion request */
            update;             /*flag to indicate presence of new data         */

    double  d,          /* range data */
            t,          /* robot's orientation angle at time of range */
            SonarTheta;  /* angle of sonar from robot center */

    POINT   posit;    /* robot's position at time of range (x, y) */
    POINT   global;   /* global position of sonar return (gx, gy) */
    POINT   SonarPosit;  /* position of sonar from center (rob_x, rob_y) */
} SONARD;


/* defines a basic segment with the start and end points, and the sonar
   it is associated with */

typedef struct  {
    POINT   start;
    POINT   end;
    double  alpha,    /* angle and length of normal from origin   */
            r;        /* to the segment                           */
    int     sonarNumber;
} SEGMENT;
```

86

```c
typedef struct  {
            /* (headx, heady, tailx, taily, sonar, alpha, r) */
    SEGMENT  seg;
            /* length of the segment                        */
    double   length;
}  LINE_SEG;


typedef struct  {      /* revised by Y. Kanayama, 07-07-93          */
    double   m00,  /* moments                                  */
             m10,
             m01,
             m20,
             m11,
             m02;
    SEGMENT  seg;  /* (startx, starty, endx, endy, n, alpha, r) */
}  CUR_DATA;


/*** Global variables  ***/
extern   int      service_flag;
extern   SONARD   sonar_table[];



/*** Prototypes  ***/

void            InitSonar(void);

double          WaitSonar(int);


   /* Interrupt handler */
void            SonarSysControl(void);


/* So the user doesn't have to include all the
   sonar header files... */

#include "sonarcard.h"
#include "sonarmath.h"
#include "sonarlog.h"

#endif
```

87

```
/******************************************************************
 *  File:       sonarmath.h                         .
 *  Comments:   12-07-94 Update to support scanning function
 *                       and new functions related to
 *                       dynamic obstacle avoidance.
 *                       Update for sonar table changes of
 *                       rob_t to SonarTheta and
 *                       offset to SonarPosit
 ******************************************************************/

#ifndef __SONARMATH_H
#define __SONARMATH_H

#include "sonar.h"

/* Types of Scanning -- Added by Jane Lochner */
#define FORWARD        0
#define BACKWARD       1
#define LEFT           2
#define RIGHT          3

/*  The following typedef was added to support the
    getBoundaries() function written as part of the
    thesis work of LCDR Lochner.   */

typedef struct {
    POINT left;
    POINT right;
        } BOUNDARY;

void      InitSonarmath(void);
void      SetSonarParameters(double, double);

double    Sonar(int);
POINT     Global(int);
LINE_SEG *GetSegment(int);
LINE_SEG *EndSegment(int);

void      CalculateGlobal(int);
void      GenerateSegment(int);

void      EnableLinearFitting(int);
void ·    DisableLinearFitting(int);
void      LinearFitting(int);


/* The following prototypes were added to support
   the new functions added in sonar.c as part of
   the thesis work of LCDR Lochner   */

POINT          ScanSonar(int,double);
BOUNDARY       getBoundaries(POINT);
double         getWidth(POINT,int);
CONFIGURATION  avoidPathVertex(double,POINT);
CONFIGURATION  avoidPathWidth(double,POINT);

#endif
```

```
/********************************************************************
 * Author      : Patrick Byrne , Yutaka Kanayama
 * Date        : 20 November 1993
 * File        : sonar.c
 * Description : Provides the global generic sonar functions
 *
 * Comments:
 *    - Fri  07-22-94  Updated for Sparc mmlll FEK
 *    - updated by Khaled morsy 11-22-94
 *    - Updated on 6 Dec 94 by Jane Lochner to add new
 *      sonar positions and to rename offset to SonarPosit
 *      and rob_t to SonarTheta in the sonar table.
 ********************************************************************/
#include "definitions.h"
#include "memsys.h"
#include "motion.h"
#include "sonarcard.h"
#include "sonarmath.h"
#include "sonarlog.h"
#include "system.h"
#include "time.h"
#include "sonar.h"

/*** Global variables ***/
int    service_flag ;
SONARD  sonar_table[NUM_SONARS];          /*one of the above struct's for each sonar  */


/* used by ServeSonar */
static const int group_array[4][4] = {
    {0, 5, 2, 7},
    {3, 4, 1, 6},
    {10, 11, 8, 9},
    {12, 13, 14, 15}
};               /* array maps sonar numbers to groups */


void
InitSonar(void)
{
    int     i;

    /* initialize sonar_table */
    for (i = 0; i < NUM_SONARS; i++)
        memset(&sonar_table[i], 0, sizeof(SONARD));

        /* set up compensation for sonar position */
    sonar_table[0].SonarTheta = 0.0;              /**/
    sonar_table[1].SonarTheta = 5.0*PI/6.0;       /**/
    sonar_table[2].SonarTheta = PI;               /**/
    sonar_table[3].SonarTheta = -PI/6.0;          /**/
    sonar_table[4].SonarTheta = PI/3.0;           /**/
    sonar_table[5].SonarTheta = PI/2.0;           /**/
    sonar_table[6].SonarTheta = -2.0*PI/3.0;      /**/
    sonar_table[7].SonarTheta = -PI/2.0;          /**/
    sonar_table[8].SonarTheta = 2.0*PI/3.0;       /**/
    sonar_table[9].SonarTheta = -5.0*PI/6.0;      /**/
    sonar_table[10].SonarTheta = -PI/3.0;         /**/
    sonar_table[11].SonarTheta = PI/6.0;          /**/
    sonar_table[12].SonarTheta = 0.0;
    sonar_table[13].SonarTheta = 1.5708;
    sonar_table[14].SonarTheta = 4.7124;
    sonar_table[15].SonarTheta = 0.0;
```

89

```c
    sonar_table[0].SonarPosit.X = 23.6;           /**/
    sonar_table[1].SonarPosit.X = -23.0;          /**/
    sonar_table[2].SonarPosit.X = -22.6;          /**/
    sonar_table[3].SonarPosit.X = 24.7;           /**/
    sonar_table[4].SonarPosit.X = 13.4;           /**/
    sonar_table[5].SonarPosit.X = 0.0;            /**/
    sonar_table[6].SonarPosit.X = -12.6;          /**/
    sonar_table[7].SonarPosit.X = 0.0;            /**/
    sonar_table[8].SonarPosit.X = -13.4;          /**/
    sonar_table[9].SonarPosit.X = -23.5;          /**/
    sonar_table[10].SonarPosit.X = 12.1;          /**/
    sonar_table[11].SonarPosit.X = 25.2;          /**/
    sonar_table[12].SonarPosit.X = 0.0;
    sonar_table[13].SonarPosit.X = 1.5708;
    sonar_table[14].SonarPosit.X = 4.7124;
    sonar_table[15].SonarPosit.X = 0.0;

    sonar_table[0].SonarPosit.Y = -0.5;            /**/
    sonar_table[1].SonarPosit.Y = 13.1;            /**/
    sonar_table[2].SonarPosit.Y = -1.0;            /**/
    sonar_table[3].SonarPosit.Y = -14.6;           /**/
    sonar_table[4].SonarPosit.Y =   21.3;          /**/
    sonar_table[5].SonarPosit.Y = 20.6;            /**/
    sonar_table[6].SonarPosit.Y = -21.3;           /**/
    sonar_table[7].SonarPosit.Y = -20.5;           /**/
    sonar_table[8].SonarPosit.Y = 21.3;            /**/
    sonar_table[9].SonarPosit.Y = -14.9;           /**/
    sonar_table[10].SonarPosit.Y = -21.3;          /**/
    sonar_table[11].SonarPosit.Y = 14.1;           /**/
    sonar_table[12].SonarPosit.Y = 0.0;
    sonar_table[13].SonarPosit.Y = 21.5;
    sonar_table[14].SonarPosit.Y = 21.5;
    sonar_table[15].SonarPosit.Y = 0.0;


    /* initialize the sonar components */
    InitSonarmath();
    InitSonarlog();

    SetSonarParameters(0.02, 5.0);
}


/******************************************************************
 * Procedure: wait_sonar(n)
 * Description:  waits in a loop until new data is available for
 * sonar n.
 ******************************************************************/

double
WaitSonar(int n)
{
    sonar_table[n].update = 0;
    while (sonar_table[n].update == 0)
       ;  /* NULL statement */
    return sonar_table[n].d;
}
```

```c
/******************************************************************
 * Procedure: serve_sonar(x,y,t,ovfl,datal,data2,data3,data4,group)
 * Description:  this procedure is the "central command" for the
 * control of all sonar related functions. It is linked with the
 * ih_sonar routine and loads sonar data to the sonar_table from
 * there. It then examines the various control flags in the
 * sonar_table to determine which activities the user wishes to take
 * place, and calls the appropriate functions. This procedure is
 * invoked approximately every thirty milliseconds by an interrupt
 * from the sonar control board.
 ******************************************************************/
void
SonarSysControl(void)
{
    static int          cnt = 0;
    int                 n;
    int                 i;
    int                 data[4];
    int                 group;
    CONFIGURATION       current;

/* overflow bit is bit 15 */
#define OVERFLOWMASK      0x8000
#define GROUP_MASK        0x18
#define SONAR_DATA0       0xfc0083f0
#define SONAR_DATA1       0xfc0083f2
#define SONAR_DATA2       0xfc0083f4
#define SONAR_DATA3       0xfc0083f6

/* blink the #1 LED */
if (++cnt > 10)   {
    cnt = 0;
    changeLEDstate(1);}

    current = getRobotConfig();

    group    = ((*(BYTE*)SONAR_CTL & GROUP_MASK) >> 3);
    data[0] = *(WORD*)SONAR_DATA0;
    data[1] = *(WORD*)SONAR_DATA1;
    data[2] = *(WORD*)SONAR_DATA2;
    data[3] = *(WORD*)SONAR_DATA3;

    for (i = 0; i < 4; i++)   {
       n = group_array[group][i];
       sonar_table[n].posit.X = current.Posit.X;
       sonar_table[n].posit.Y = current.Posit.Y;
       sonar_table[n].t = current.Theta;

           /*   -1 was returned if there was no echo */
       if (data[i] & OVERFLOWMASK)
           sonar_table[n].d = INFINITY;

       else
       (   /* only first 12 bits are data, so mask the data */
           data[i] &= 0xfff;
           sonar_table[n].d = data[i] * 0.1029; }

       CalculateGlobal(n);
       if (sonar_table[n].fitting == 1)
               LinearFitting(n);

       /* log the data for this sonar */
       LogSonarData(n);
           }                                    91
}
```

```c
/**************************************************************
 * Author      : Patrick Byrne
 * Date        : 20 November 1993
 * File        : sonarmath.c
 * Description : Provides the main sonar functions
 *
 * Comments:
 *    Fri  07-22-94  Updated for Sparc mml11 FEK
 *    Wed  12-07-94  Five new functions added by
 *                   Jane Lochner to support scanning
 *                   and dynamic obstacle avoidance
 **************************************************************/

#include "definitions.h"
#include "sonar.h"
#include "stdiosys.h"
#include "math.h"
#include "motion.h"
#include "memsys.h"
#include "sonarlog.h"
#include "sonarmath.h"

#define QMAX   50
#define SONARS 11

/*** Local variables ***/
static double      C1, C2;

static LINE_SEG    Queue[SONARS][QMAX];
static int         Head[SONARS];
static int         Tail[SONARS];
static int         Empty[SONARS];

static LINE_SEG    segstruct;      /* temporary storage for get_segment func. */

static int         SegListHead[NUM_SONARS];   /*points to oldest segment array element  */
static int         SegListTail[NUM_SONARS];   /*points to newest segment array element  */

static LINE_SEG    seg_list[NUM_SONARS][5];       /*segments for working memory
     */
static CUR_DATA    segment_data[NUM_SONARS];    /*interim data for all sixteen sonars */

/*** Global variables  ***/

/*** Local Prototypes  ***/
void     Enqueue(int, LINE_SEG*);
void     AddToSegment(int, POINT);
void     ResetMoments(int);
void     BuildList(LINE_SEG*, int);


/*** Code ***/
void
InitSonarmath(void)
{
   int  i;

   for (i = 0; i < NUM_SONARS; i++) {
      ResetMoments(i);
      memset(&segment_data[i], 0, sizeof(CUR_DATA));

      Empty[i] = TRUE;
      Head[i] = 1;
      Tail[i] = 1;                      92
   }
}
```

```c
/*******************************************************************
 * Procedure: set_sonar_parameters(cl,c2)
 * Description:  allows the user to
 * adjust constants which control the  linear fitting algorithm. C1 is
 * a multiplier to allow more lenancy for greater sonar ranges.
 * C2 is an absolute value; both are used to determine if an
 * individual data point is usable for the algorithm.  Default values
 * are set in main.c to .02, 5.0 respectively.
 *******************************************************************/
void
SetSonarParameters(double cl, double c2)
{
    Cl = cl;
    C2 = c2;
}



/*******************************************************************
 * Procedure:  sonar(n)
 * Description:  returns the distance (in
 * centimeters) sensed by the  n_th ultrasonic sensor.  If no echo is
 * received, then INFINITY(1.0e6) is  returned.  If the distance is less than 10
 * cm, then a 0 is returned.
 *******************************************************************/
double
Sonar(int n)
{
    return sonar_table[n].d;
}



/*******************************************************************
 * Procedure: global(n)
 * Description: returns a structure of type
 * posit containing the global x and y coordinates of the position of
 * the last sonar return.
 *******************************************************************/
POINT
Global(int n)
{
    return  sonar_table[n].global;
}

/*************************************************
void Enqueue()
This function is called by generate_segment(). The
sonar number and newest line_segment for that sonar
are passed int. It simply places the latest segment
produce by a sonar with linear_fitting and places it
into a circular queue.
*************************************************/
void
Enqueue(int i, LINE_SEG *Seg)
{
    int         j;

    if (Head[i]==Tail[i] && Empty[i] == FALSE)
        printf("Sonar segment queue is Full");
    else {
        j = Tail[i];
        Queue[i][j] = *Seg;

        Tail[i] = 1 + (Tail[i] % QMAX);
        Empty[i] = FALSE;                    93
    }
}
```

```
/*******************************************************
LINE_SEG get_segment(sonar)
returns the pointer to the oldest completed unread
segment of the sonar passed in. If there is no completed
unread segment NULL is returned.
********************************************************/
LINE_SEG *
GetSegment(int i)
{
    LINE_SEG    *Current_seg;
    int         j;

    if (Empty[i])
        Current_seg = NULL;
    else  {
        j = Head[i];
        Current_seg = &Queue[i][j];
        Head[i] = 1 + (Head[i] % QMAX);
        if (Head[i] == Tail[i])
            Empty[i] = TRUE;
    }.
    return Current_seg;
}




/*****************************************************************************
 * Procedure: EndSegment(n)
 * Description: this procedure allocates
 * memory for the segment data structure, loads the correct values
 * into it and returns a pointer to the structure.
 *****************************************************************************/

LINE_SEG *
EndSegment(int n)
{
    SEGMENT         tmpSeg;
    LINE_SEG        *seg_ptr;
    double          length, delta;

    seg_ptr = &segstruct;

    tmpSeg = segment_data[n].seg;
    delta = tmpSeg.start.X * cos(tmpSeg.alpha) +
            tmpSeg.start.Y * sin(tmpSeg.alpha) - tmpSeg.r;
    tmpSeg.start.X -= delta * cos(tmpSeg.alpha);
    tmpSeg.start.Y -= delta * sin(tmpSeg.alpha);
    delta = tmpSeg.end.X * cos(tmpSeg.alpha) +
            tmpSeg.end.Y * sin(tmpSeg.alpha) - tmpSeg.r;
    tmpSeg.end.X -= delta * cos(tmpSeg.alpha);
    tmpSeg.end.Y -= delta * sin(tmpSeg.alpha);
    length = sqrt(SQR(tmpSeg.start.X - tmpSeg.end.X) + SQR(tmpSeg.start.Y - tmpSeg.end.Y))

    seg_ptr->seg = tmpSeg;
    seg_ptr->length = length;
    seg_ptr->seg.sonarNumber = n;

    return seg_ptr;
}
```

94

```c
/********************************************************************
 * Procedure: CalculateGlobal(n)
 * Description: this procedure
 * calculates the global x and y coordinates for the range value and
 * robot configuration in the sonar table. The results are stored in
 * the sonar table.
 ********************************************************************/
void
CalculateGlobal(int n)
{
    double    lx, ly, lt, range, SonarTheta, rob_x, rob_y;
    CONFIGURATION global;

    range = sonar_table[n].d;
    if (range >= INFINITY0)  {
        sonar_table[n].global.X = INFINITY;
        sonar_table[n].global.Y = INFINITY;
    }
    else {
        rob_x = sonar_table[n].SonarPosit.X;
        rob_y = sonar_table[n].SonarPosit.Y;
        SonarTheta = sonar_table[n].SonarTheta;
        global = getRobotConfig();

        /* vehicle compose sonar */
        lx = global.Posit.X + (cos(global.Theta) * rob_x) -
                              (rob_y * sin(global.Theta));
        ly = global.Posit.Y + (sin(global.Theta) * rob_x) +
                              (rob_y * cos(global.Theta));
        lt = SonarTheta + global.Theta;

        /* vehicle compose sonar range */
        sonar_table[n].global.X = lx + (cos(lt) * range);
        sonar_table[n].global.Y = ly + (sin(lt) * range);
    }
}

/********************************************************************
 * Procedure: add_to_segment(n, x, y)  * Description: this procedure
 * calculates new interim data for the line segment and stores it in
 * segment_data[n]. It also changes the end point values to the point
 * being added.
 ********************************************************************/
void
AddToSegment(int n, POINT p)
{
    double    m00, m10, m01, m20, m11, m02;
    double    alpha, r;
    double    mux, muy, mm20, mm11, mm02;

    m00 = segment_data[n].m00 += 1.0;
    m10 = segment_data[n].m10 += p.X;
    m01 = segment_data[n].m01 += p.Y;
    m20 = segment_data[n].m20 += SQR(p.X);
    m11 = segment_data[n].m11 += p.X * p.Y;
    m02 = segment_data[n].m02 += SQR(p.Y);

    if (m00 < 1.5)
        segment_data[n].seg.start = p;

    mux = m10 / m00;
    muy = m01 / m00;
    mm20 = m20 - SQR(m10) / m00;
    mm11 = m11 - m10 * m01 / m00;
    mm02 = m02 - SQR(m01) / m00;
```

95

```c
    if (m00 > 1.5)   {
        alpha = atan2(-2.0 * mm11, (mm02 - mm20)) / 2.0;
        r = mux * cos(alpha) + muy * sin(alpha);

        segment_data[n].seg.alpha = alpha;
        segment_data[n].seg.r = r;
        segment_data[n].seg.end = p;
    }
}


/*****************************************************************************
 * Procedure: reset_moments(n);
 * Description: resets the accumulative
 * values in segment_data[n]  (m00,m10,m01,m20,m11,m02) to zero.
 *****************************************************************************/

void
ResetMoments(int n)
{
    segment_data[n].m00 = 0.0;
    segment_data[n].m10 = 0.0;
    segment_data[n].m01 = 0.0;
    segment_data[n].m20 = 0.0;
    segment_data[n].m11 = 0.0;
    segment_data[n].m02 = 0.0;
}


/*****************************************************************************
 * Procedure: generate_segment(n)
 * Description: this function
 * completes segments at the end of a data run. Necessary because the
 * linear fitting function only terminates a segment based on the data
 * - it has no way of knowing that the user has stopped collecting data.
 *****************************************************************************/

void
GenerateSegment(int n)
{
    LINE_SEG        *seg_ptr;

    if (segment_data[n].m00 > 10.0)   {
        seg_ptr = EndSegment(n);
        BuildList(seg_ptr, n);
        Enqueue(n,seg_ptr);
    }
    ResetMoments(n);
}


/*****************************************************************************
 * Procedure: EnableLinearFitting(n)
 * Description:  causes the background system to gather data points
 *      from sonar n and form them into line segments as governed by
 *      the linear fitting algorithm.
 *****************************************************************************/

void
EnableLinearFitting(int n)
{
    sonar_table[n].fitting = 1;
}
```

96

```
/**********************************************************************/
/* Procedure:  DisableLinearFitting(n)
 * Description:  causes background system to cease forming line
 * segments for sonar n.
 **********************************************************************/

void
DisableLinearFitting(int n)
{
   GenerateSegment(n);
   sonar_table[n].fitting = 0;

}




/**********************************************************************
 * Procedure: LinearFitting(n)
 *            Revised by Y. Kanayama,07-07-93
 * Description: this procedure controls the fitting of point
 * data to straight line segments. First it tests if the new coming
 * point is not far from the fitted line.  If the test is passed, the
 * point is added to test if the thinness test is passed.  If it is
 * passed, the addition is finalized. If any of the tests fail,
 * the line segment is ended and a new one started. The completed line
 * segment is stored in a data structure called segment, and segments
 * are linked together in a linked list.
 **********************************************************************/

void
LinearFitting(int n)
{
   POINT           p;
   double          m00;
   double          alpha, r, delta;
   double          sonar_range;
   LINE_SEG        *finished_segment;

   sonar_range = sonar_table[n].d;
   if (sonar_range < 9.3 || sonar_range > 409.0) {
      GenerateSegment(n);
      return;
   }

   p = sonar_table[n].global;  /* temporary moments          */
   m00 = segment_data[n].m00;

   if (m00 < 1.5)  {
      AddToSegment(n, p);
      return;
   }

   r = segment_data[n].seg.r;
   alpha = segment_data[n].seg.alpha;
   delta = fabs(r - p.X * cos(alpha) - p.Y * sin(alpha));

   if (delta > MAXVAL(C2, C1 * sonar_range))
      GenerateSegment(n);

   AddToSegment(n, p);
   return;
}                                 /* end linear_fitting */
```

97

```c
/***************************************************************************
 * Procedure: build_list(ptr, n);                    .
 * Description: this function accepts
 * a pointer to a segment data structure and a sonar number, and
 * appends the segment structure to the tail of a linked list of
 * structures for that sonar.
 ***************************************************************************/
void
BuildList(LINE_SEG *ptr, int n)
{
    int             next;

    if (SegListTail[n] == -1)
        SegListHead[n] = 0;
    next = (SegListTail[n] < 4) ? ++SegListTail[n] : 0;

    if (next == SegListHead[n])
        SegListHead[n] = (SegListHead[n] < 4) ? ++SegListHead[n] : 0;
    seg_list[n][next] = *ptr;
    LogSonarSegmentData(n, seg_1ist[n][next]);
}


/***************************************************************************
 * Procedure:    ScanSonar(int dir, double dist)
 * Description: Function allows user to scan in one of four
 *              directions for obstacles.  Function will return
 *              when it detects an obstacles within the specified
 *              distance. Default is forward scan.
 * Inputs:      Scan Direction and detection distance
 * Outputs:     Global coordinates of obstacle
 * Date:        06 DEC 94
 * Author:      LCDR Jane Lochner
 ***************************************************************************/
POINT
ScanSonar(int dir,double dist)             .
{
  double hit1=9999.9,hit2=9999.9,hit3=9999.9;
  POINT obstacle;
  int sonar1,sonar2,sonar3;

  switch(dir)   {
    case FORWARD:
     .sonar1=S330;
      sonar2=S000;
      sonar3=S030;
      break;
    case BACKWARD:
      sonar1=S210;
      sonar2=S180;
      sonar3=S150;
      break;
    case LEFT:
      sonar1=S300;
      sonar2=S270;
      sonar3=S240;
      break;
    case RIGHT:
      sonar1=S060;
      sonar2=S090;
      sonar3=S120;
    default:
      sonar1=S330;
      sonar2=S000;
      sonar3=S030;
      break;
    }
```

98

```
        EnableSonar(sonar1);
        EnableSonar(sonar2);
        EnableSonar(sonar3);
        do     {
                waitMS(30);
                hit1=Sonar(sonar1);
                waitMS(30);
                hit2=Sonar(sonar2);
                waitMS(30);
                hit3=Sonar(sonar3);
                } while ((hit1>dist || hit1<1.0) &&
                         (hit2>dist || hit2<1.0) &&
                         (hit3>dist|| hit3<1.0));

        if (hit1<dist)
            obstacle=Global(sonar1);
        if (hit2<dist)
            obstacle=Global(sonar2);
        if (hit3<dist)
            obstacle=Global(sonar3);

        DisableSonar(sonar1);
        DisableSonar(sonar2);
        DisableSonar(sonar3);

        return(obstacle);
}




/*****************************************************************
*   Procedure:   getBoundaries(POINT scan)
*   Description: Function returns the left and right
*               boundaries of an object.  This is a
*               dummy function to be implemented at
*               a later date.  The user just inputs
*               values to be returned.
*   Date:        06 DEC 94
*   Author:      LCDR Jane Lochner
*****************************************************************/
BOUNDARY
getBoundaries(POINT scan)
{
BOUNDARY obstacle;

obstacle.left.Y = 105.0;
obstacle.left.X = 150.0;
obstacle.right.Y = 25.0;
obstacle.right.X = 150.0;

return(obstacle);
}
```

99

```
/*********************************************************
 *  Procedure:  getWidth (POINT scan,int direction)
 *  Description:  Function returns the width of object
 *                on the designated side.  This is a
 *                dummy function to be implemented at
 *                a later date. The user just inputs
 *                the values for testing purposes.
 *  Date:         06 DEC 94
 *  Author:       LCDR Jane Lochner
 *********************************************************/
double
getWidth(POINT scan,int direction)
{
  double width;

  if (direction == LEFT)
    width = 80.0;
  else
    width = -55.0;
  return (width);
}


/*********************************************************
 *  Procedure: avoidPathVertex(double safety, POINT scan)
 *  Description:  Calculates path to avoid obstacle
 *                using designated safety margin and the
 *                outer vertices of the object
 *  Inputs:   safety margin, global coordinates of closest
 *            point
 *  Outputs: New path to avoid obstacle
 *  Date:     06 DEC 94
 *  Author:   LCDR Jane Lochner
 *********************************************************/
CONFIGURATION
avoidPathVertex(double safety, POINT scan)
{
CONFIGURATION Current,avoid;
POINT left,right;
BOUNDARY obstacle;
double LeftDist,RightDist;

Current = getRobotConfig();
obstacle = getBoundaries(scan);

LeftDist = -(obstacle.left.X-Current.Posit.X)*sin(Current.Theta)+
           (obstacle.left.Y-Current.Posit.Y)*cos(Current.Theta)+safety;
RightDist = -(obstacle.right.X-Current.Posit.X)*sin(Current.Theta)+
           (obstacle.right.Y-Current.Posit.Y)*cos(Current.Theta)-safety;

left.X = Current.Posit.X - LeftDist*sin(Current.Theta);
left.Y = Current.Posit.Y + LeftDist*cos(Current.Theta);
right.X = Current.Posit.X - RightDist*sin(Current.Theta);
right.Y = Current.Posit.Y + RightDist*cos(Current.Theta);


if (LeftDist > -RightDist)  {
        avoid.Posit.X = right.X;
        avoid.Posit.Y = right.Y;}
else {
        avoid.Posit.X = left.X;
        avoid.Posit.Y = left.Y;}
avoid.Theta = Current.Theta;
avoid.Kappa = Current.Kappa;
return(avoid);
}
```

```
/***************************************************************
*    Procedure: avoidPathWidth(double safety, POINT scan)
*    Description:  Calculates path to avoid obstacle
*                  using designated safety margin and the
*                  left and right widths of the object
*    Inputs:   safety margin, global coordinates of closest
*              point
*    Outputs: New path to avoid obstacle
*    Date:      06 DEC 94
*    Author:   LCDR Jane Lochner
****************************************************************/
CONFIGURATION
avoidPathWidth(double safety, POINT scan)
{
CONFIGURATION Current,avoid;
POINT left,right;
double LeftDist,RightDist;

Current = getRobotConfig();
LeftDist = getWidth(scan,LEFT);
RightDist = getWidth(scan,RIGHT);

left.X = Current.Posit.X - LeftDist*sin(Current.Theta);
left.Y = Current.Posit.Y + LeftDist*cos(Current.Theta);
right.X = Current.Posit.X - RightDist*sin(Current.Theta);
right.Y = Current.Posit.Y + RightDist*cos(Current.Theta);


if (LeftDist > -RightDist)
        {
         avoid.Posit.X = right.X;
         avoid.Posit.Y = right.Y;
                           }
else
            {
         avoid.Posit.X = left.X;
         avoid.Posit.Y = left.Y;
                           }
avoid.Theta = Current.Theta;
avoid.Kappa = Current.Kappa;
return(avoid);
}
```

101

LIST OF REFERENCES

Byrne, P. G., *A Mobile Robot Sonar System with Obstacle Avoidance*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1994.

Davis, E., *Representations of Commonsense Knowledge,* Morgan Kaufmann Publishers, Inc., 1990.

Elfes, A., "Sonar-Based Real-World Mapping and Navigation," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 3, pp. 249-265, June 1987.

Kinsler, L.E. and others, *Fundamental of Acoustics*, John Wiley & Sons, 1982.

Michuie, M., *Research on the Sonar Hardware System on an Autonomous Mobile Robot*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1994.

Olson, H. F., *Elements of Acoustical Engineering*, D. Van Nostrand Company, Inc., 1947.

Saleh, B. E. A., and Teich, M. C., *Fundamentals of Photonics*, John Wiley & Sons, 1991.

Sherfey, S., *A Mobile Robot Sonar System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1991.

Yamabico Manual, Naval Postgraduate School, Monterey, California, 12 April 1993.

BIBLIOGRAPHY

Beranek, L. L., *Acoustics*, McGraw-Hill Book Company, 1954.

Kanayama, Y., "Mathematical theory of Robotics: Introduction to 2D Spatial Reasoning," *Lecture Notes of the Advanced Robotics Course*, Department of Computer Science, Naval Postgraduate School, Winter Quarter 1994.

Kanayama, Y. Noguchi, T. and Hartman, B., "Sonar Data Interpretation for Autonomous Mobile Robots," Naval Postgraduate School.

McLachlan, N. W., *Loud Speakers -- Theory, Performance, Testing and Design*, Dover Publications, Inc., 1960.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52 2
   Naval Postgraduate School
   Monterey, California 93943-5101

3. Chairman, Code PH/Cw 1
   Physics Department
   Naval Postgraduate School
   Monterey, California 93943-5000

4. Dr. Yutaka Kanayama, Code CS/Ka 2
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5118

5. Dr. Donald Walters, Code PH/We 1
   Physics Department
   Naval Postgraduate School
   Monterey, California 93943-5000

6. Dr. Donald Brutzman, Code OR/Br 1
   Operations Research Department
   Naval Postgraduate School
   Monterey, California 93943-5000

7. Dr. Xiaoping Yun, Code EC/YX 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

8. Mr. Michael Williams 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, California 93943-5000