(1)

# NAVAL POSTGRADUATE SCHOOL
# Monterey, California

THESIS

# RESEARCH ON THE SONAR HARDWARE SYSTEM ON AN AUTONOMOUS MOBILE ROBOT

by

Masakuni Michiue

June 1994

Thesis Advisor: Yutaka Kanayama

Approved for public release; distribution is unlimited.

# REPORT DOCUMENTATION PAGE

*Form Approved OMB No. 0704-0188*

| | | |
|---|---|---|
| 1. AGENCY USE ONLY (Leave Blank) | 2. REPORT DATE June 1994 | 3. REPORT TYPE AND DATES COVERED Master's Thesis |

| | |
|---|---|
| 4. TITLE AND SUBTITLE Research on the Sonar Hardware System on an Autonomous Mobile Robot | 5. FUNDING NUMBERS |
| 6. AUTHOR(S) Michiue, Masakuni | |
| 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 United States Government.

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

13. ABSTRACT *(Maximum 200 words)*

The autonomous mobile robot, Yamabico-11, recognizes distance from obstacles by transmit and receive sonar pair. However, the current sonar amplification has not been enough to obtain reliable range information. This thesis describes to improve the sonar analog circuits on the autonomous mobile robot so that they obtain more robust range information.

One improvement was a change in the driving voltage of the transmit transducer from 5 volts to 12 volts which doubles the strength of sonar signal received by the pickup sensor. After changing the voltage source, it was found that there was spillover leakage directly from the transmitter to the receiver transducer. The amplifier sensitivity was decreased for the first one millisecond to reduce the spillover.

| | |
|---|---|
| 14. SUBJECT TERMS Autonomous mobile robot, sonar hardware system | 15. NUMBER OF PAGES 65 |
| | 16. PRICE CODE |

| | | | |
|---|---|---|---|
| 17. SECURITY CLASSIFICATION OF REPORT Unclassified | 18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified | 19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified | 20. LIMITATION OF ABSTRACT Unlimited |

# RESEARCH ON THE SONAR HARDWARE SYSTEM ON AN AUTONOMOUS MOBILE ROBOT

by

Masakuni Michiue
Lieutenant Commander, Japanese Navy
B.S., National Defense Academy, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

June 1994

Author: Masakuni Michiue
Masakuni Michiue

Approved By: [illegible]
Yutaka Kanayama, Thesis Advisor

David C. Jenn
David C. Jenn, Second Reader

[illegible]
Michael A. Morgan, Chairman,
Department of Electrical & Computer Engineering

# ABSTRACT

The autonomous mobile robot, Yamabico-11, recognizes distance from obstacles by a transmit and receive sonar pair. However, the current sonar amplification has not been enough to obtain reliable range information. This thesis describes methods to improve the sonar analog circuits on the autonomous mobile robot so that they obtain more robust range information.

One improvement was a change in the driving voltage of the transmit transducer from 5 volts to 12 volts which doubles the str[illegible] sonar signal received by the pickup sensor. After changing the voltage source, it was found that there was spillover leakage directly from the transmitter to the receiver transducer. The amplifier sensitivity was decreased for the first one millisecond to reduce the spillover.

# TABLE OF CONTENTS

# LIST OF FIGURES

# I. INTRODUCTION

## A. BACKGROUND

Recent progress in robotics has been remarkable. Efforts to increase robot intelligence have been investigated mainly from the software side. On the other hand, the hardware systems of robots also have become more capable due to the advancement of electronics and mechanics. Finally, sensor technology is progressing rapidly allowing the sensors to be more compact, sensitive, and precise.

The autonomous mobile robot, Yamabico-11, at the Naval Postgraduate School, is used to demonstrate new technology in each of the above areas. Particularly the sonar hardware system, which is used to recognize a distance, has to be improved because of unreliable range information due to noise. Thus the sonar hardware system must be replaced or redesigned to obtain more robust range information.

## B. OVERVIEW

Currently, the Yamabico-11 has a sonar system connected to the VME system bus via an Omnibyte VME interface board. A CPU controls the sonar ranging system through the VME bus to read the range data for transmittal the main program. Each sonar sensor is controlled by a 8748 controller and detects distance from an obstacle. The sonar driver board serves as an interface between the sensor and controller/range counter. Sonar data detected by the sensor are stored and sent to the Omnibyte VME interface board.

The sonar analog system consists of twelve transmit/receive sensor pairs and three sonar driver boards. It is this portion of the system that will be improved; replacing all transducers and redesigning the sonar driver board to obtain more robust range data. To achieve this the driving voltage of the transmit transducer was increased from 5 volts to 12 volts, which doubles the strength of sonar signal received by the pickup sensor.

After changing the voltage source, it was found that the transmit-receive isolation was not adequate. This leakage problem was eliminated by decreasing the amplifier sensitivity for the first one millisecond. This method did not affect valid sonar returns.

Chapter II describes the sonar system used on the Yamabico-11 and the operation of the current circuits. The principle of range finding is discussed briefly. In Chapter III the analog portion of the sonar is described and several methods of improving range performance are investigated. The redesign of the circuits is presented in Chapter IV. Finally, Chapter V summarizes the accomplishments and includes recommendations for future work.

# II. SONAR SYSTEM DESCRIPTION

## A. OVERVIEW

The Yamabico's sonar system includes a dedicated sonar board with a microprocessor which controls the sonar transducers. The robot's central processing unit is interrupted only when data becomes available from the sonar array. This architecture enables parallel computation cf sonar tasks with other CPU tasks. The sonar hardware system controls the robot's array of sonar range finders. A photo of the Yamabico-11 is given in Figure 1. The transmit/receive cones are visible at each corner of the support frame. Figure 2 shows the current hardware configuration.

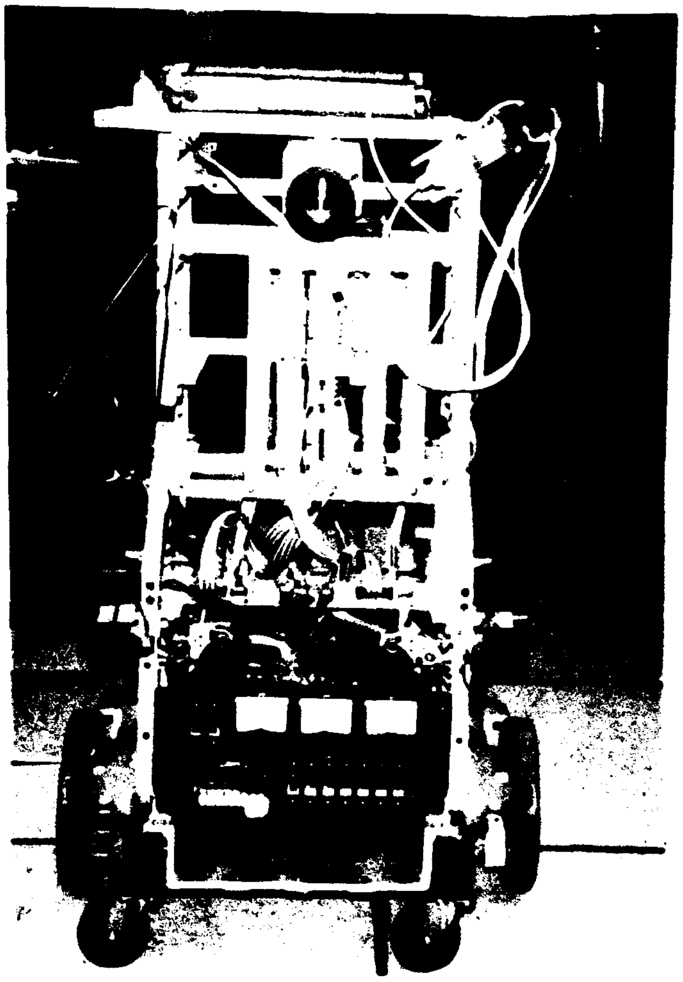


**Figure 1: The Autonomous Mobile Robot (Yamabico-11)**

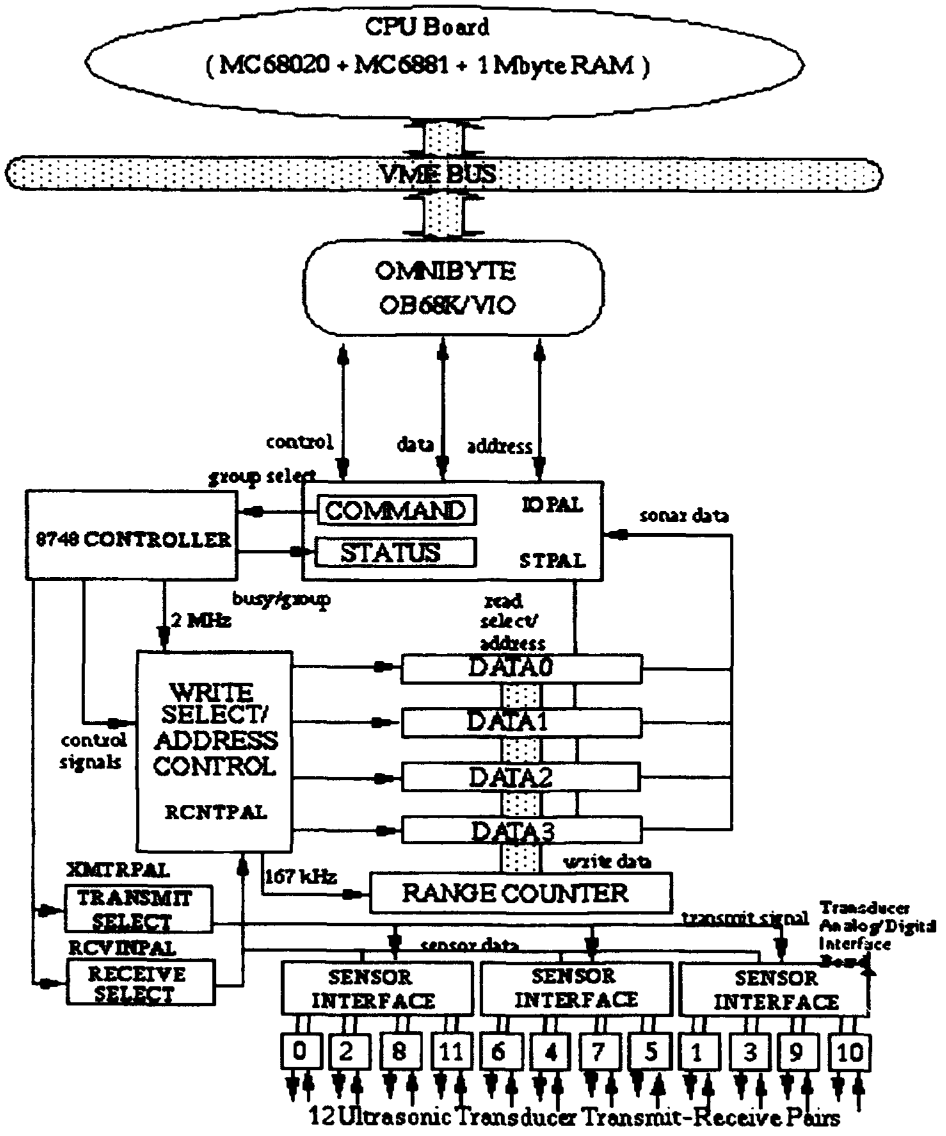


**Figure 2: Sonar Hardware Architecture**

## B. PRINCIPLE OF RANGE FINDING

There are four 16 bit data registers on the sonar control board, one for each in a logical group. When the sonar transmit pulse is sent, a pulse is sent to the driver/amplifier boards and a counter is then started which increments each of the data registers every 6 microseconds. This round trip time period is equivalent to a range

$$range = (velocity\ of\ sound) \times (time)/2 \qquad (3.1)$$

$$range = (340000\ millimeter/second) \times (6\ microsecond)/2 = 1.02\ millimeter \qquad (3.2)$$

The incrementing of a particular data register continues until an echo is received or the range gate times out. The first 12 bits of the data register are allotted for range accumulation, thus allowing a maximum range of 4.177 meters. If the range gate should time out before an echo is received, the high bit of the over ranged sonar's data register is set to 1. This is the "over range" bit and is used to signal the ensuing software that no echo was received. Bits 12, 13, and 14 of the data registers are not used. When the ranging cycle is complete, the appropriate group number is written into bits 4 and 5 of the status register and the "ready" bit, bit 7 of the status register, is set to 1. The ready bit is used as a flag when operating in the polled mode i.e. without interruptions.

The maximum range measurement is 4 meters. The data resolution is 1 millimeter. The sonar transducers operate at a frequency of 40 kHz. This is (1/40 kHz) = 25 microseconds per cycle. Assuming the speed of sound is 340 meters/second in air at sea level, the round-trip time is

$$time = \frac{400\ centimeter}{34000\ centimeter/second} \times 2 = 23.53\ milliseconds \qquad (3.3)$$

This round trip time is the period in which a valid echo may be received and is referred to as the *receive gate*. This time interval is set to to 24 milliseconds, a number derived by division of the sonar system's 2 MHz clock pulse to ensure that the receiver is not falsely triggered by a direct path reception from it's adjacent transmitter. Receivers are disabled until the transmit pulse is completed. The disadvantage of this eclipsing is a minimum measurable range equal to half the distance sound would travel in the time of a transmit pulse.

The sampling rate can be as high as 41 Hz with only one group enabled (based on a 24 millisecond read gate as determined in Equation 3.2) and will be halved for each additional group enabled. At a nominal robot speed of 30 centimeters per second, this sampling rate provides an updated range within 0.75 centimeter of travel, exceeding our desired positional accuracy of 1 centimeter. Of course, real performance will be affected by any delay in reading the data registers due to other demands on the central processor (processing the sonar data, controlling motion, etc.).

The minimum range of detection is

$$range = (34000\ centimeter/second) \times (1\ millisecond)/2 = 17\ centimeters \tag{3.4}$$

This minimum range lies approximately 9 centimeters outside the periphery of the robot. In order to allow the measurement of objects up to the periphery of the robot, the pulse width was decreased to 0.5 milliseconds which reduced the minimum range to 8.5 centimeters.

In practice, the minimum range is set to 9.6 centimeters because of the firmware; the additional distance is due to the time needed for switching and settling in the circuitry.

## C. SONAR GROUPING

In order to reduce sampling time, the sonars are operated in logical groups of four. All sonars of a logical group are pulsed simultaneously. The group to be fired is determined by the value of the corresponding bit in the *command resister* of the sonar control board, which is the user set with an Model-Based Mobile Robot Language (MML) function (Figure 2). Hence, if bit 2 is set to 1, group 2 sonars will be pulsed. When more than one group is selected, the sonar control board will trigger one group at one time in a sequential fashion. The sensors of a logical group are pulsed simultaneously and thus the sampling time is reduced by a factor of four as compared to firing the sensors individually. The sonars of each logical group are oriented in such a way as to

1. prevent mutual interface
2. provide a "look" in all four directions, and
3. present a similar aspect from each sonar during a rotational scan.

Logical group 0 consists of sonars 0, 2, 5, and 7; group 1 consists of sonars 1, 3, 4, and 6; group 2 consists of sonars 8, 9, and 11; and group 3 is a "virtual" group which consists of four permanent test values. The sonars of a group are symmetric about the robot's axis of rotation. Figure 3 shows the Yamabico sonar placement.

In addition to being grouped *logically*, the sonars are also grouped *physically*. The sonars are physically grouped so as to distribute the electrical load over the driver boards evenly and to minimize any electrical transients associated with operation of the sonar. The physical grouping connects sonars 0, 2, 8, and 11 to drive/amplifier board 1; sonars 4, 5, 6, and 7 to board 2; and sonars 1, 3, 9, and 10 to board 3. The reader will note that pairs of sonars from logical groups are assigned to physical groups.

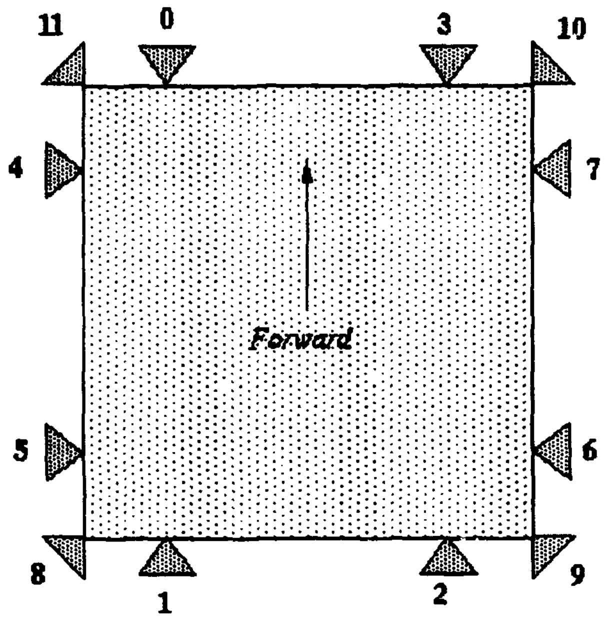


**Figure 3: Yamabico Sonar Placement**

## D. OPERATION BY LOCAL CONTROLLER

The sonar control board is actually a daughtercard which rides on a VME bus mothercard. The mothercard carries address decoders, bus drivers and interrupt control circuitry in the Bus Interface Module (BIM).

When the sonar has completed a ranging cycle an interrupt request is provided to the BIM. The BIM's *control register* holds information which determines whether an interrupt is to be generated or not, and if so, which interrupt level is to be generated. Presuming an interrupt is generated, when the correct acknowledgment returns on the address lines the BIM's *vector register* provides the vector table entry to the central processor and finds the vector to the interrupt handler. The correct interrupt level, the interrupt enable bit and interrupt vector are loaded to the BIM during software initialization.

Each of the data resisters is individually addressed on the VME bus by a VME short address, as is the status register. Transfer of the data is straightforward. The interrupt handler simply reads the correct register, masks out the unwanted bits and writes the data to the stack. When the last data resister is read, the sonar system resets the data registers and commences a ranging cycle on the next sonar group in its sequence. The system will continue to operate autonomously until all the sonars are disabled.

# III. SONAR ANALOG SYSTEM IMPROVEMENT

## A. CHANGING OF SUPPLY VOLTAGE

Each sonar consists of a transmit and receive transducer pair (Figure 4) which are connected to a sonar driver board. Each sonar driver board in turn handles four transducer pairs. The board provides a TTL level interface to each transducer. The transmit signal to the sonar driver board is a TTL high when the transmitter is inactive. A 40 kHz active low square wave must be provided to generate the transmit ping. There is a separate drive line for each of the four sonar transmitters. The transmit transducers are driven by an SGS L293 integrated circuit. This chip contains four separate buffer drivers. Each buffer driven by TTL level signals from separate 74LS240 inverters drives one sonar transducer.

The drive level to the transmit transducers was originally a +5 volt square wave. In order to increase the output voltage of transmit transducer, the equivalent of a 40 kHz, + 10 volt square wave circuit was achieved by using a 2 MHz clock pulse and TTL chips. Figure 5 shows an experimental circuit for transmitters.

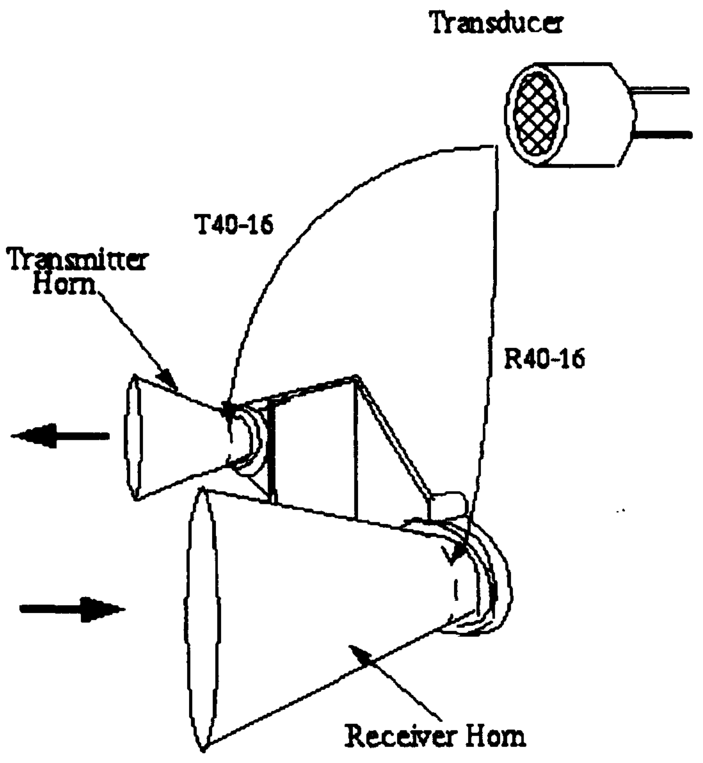


**Figure 4: Sonar Sensor Pair**

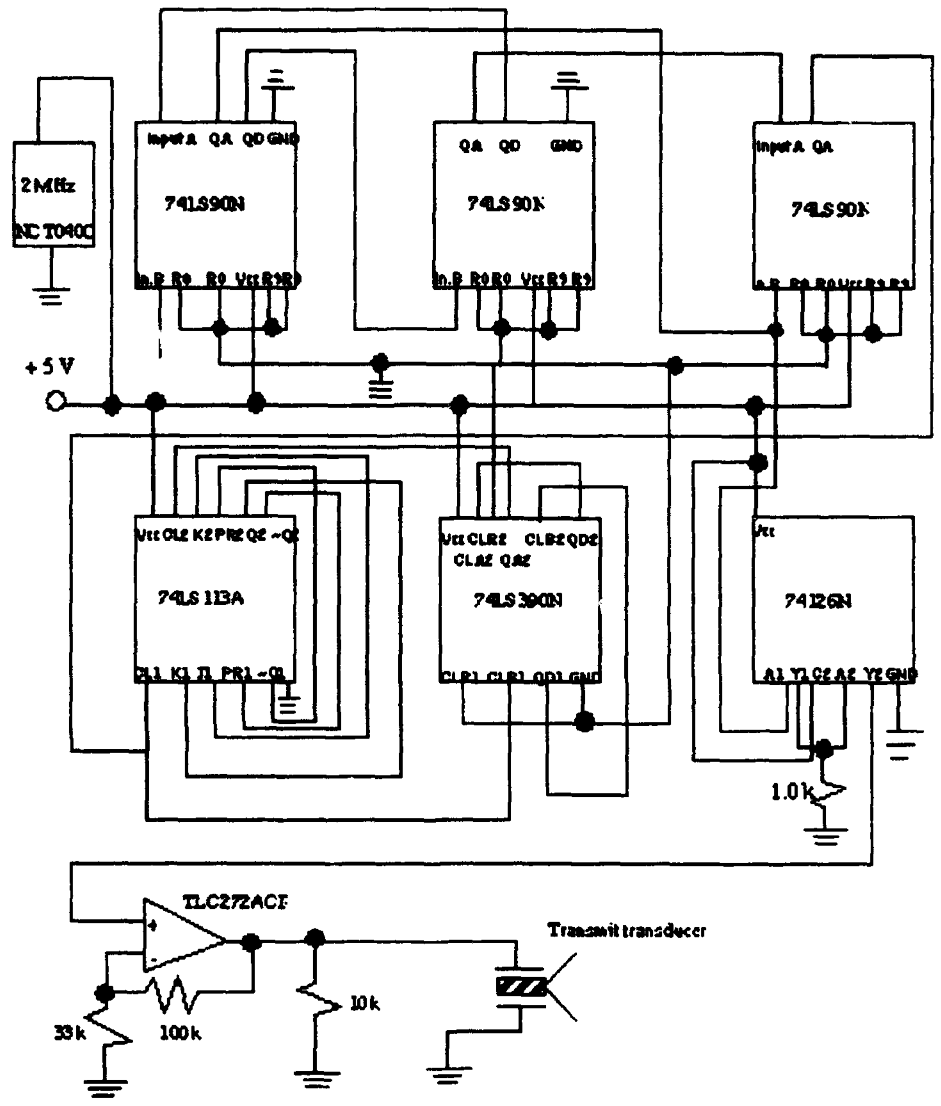


Figure 5: Experimental Circuit for Transmitter

## B. SPILLOVER REDUCTION

The current receiver amplifier is a TLC272ACP dual integrated operational amplifier. The amplifiers in this chip are designed to operate from a single supply voltage of +5 volts. The two stages are connected to form a DC amplifier with a gain of 3300. The amplifier output runs to a 74LS14 schmitt trigger, and then to an output driver.

Driving the transmit transducer with + 10 volts causes a serious transmit spillover problem: direct signals from the transmitter to the receiver mask valid returns. To suppress the transmit spillover while still allowing for valid sonar returns, it is necessary to modify the receive amplifiers. Circuitry is added that decreases sensitivity for the first millisecond after the end of the transmit pulse. Figure 6 shows the spillover reduction circuit based on the original circuit with adding of a capacitance-resistance circuitry and supply voltage of - 5 volts and -12 volts.

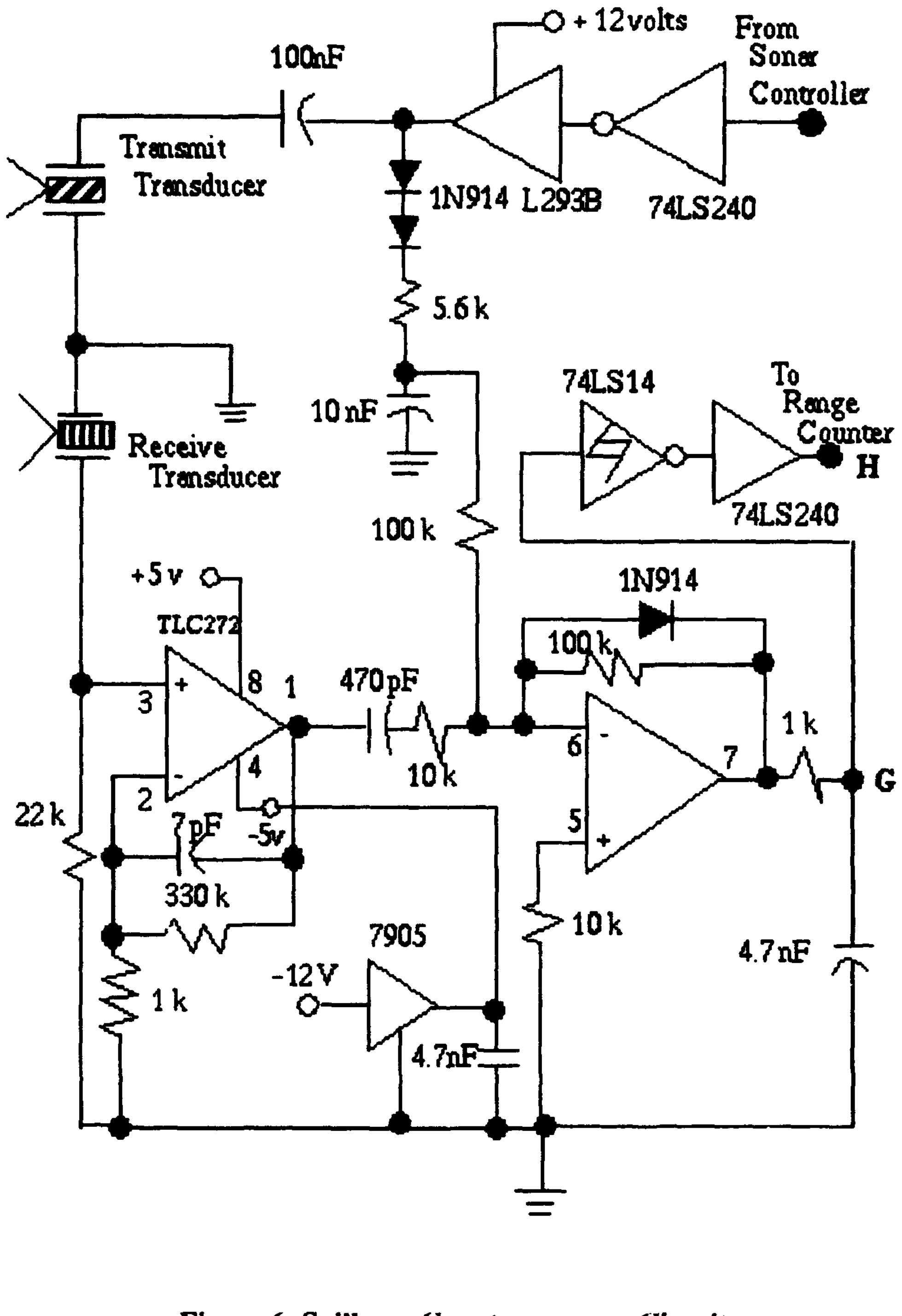


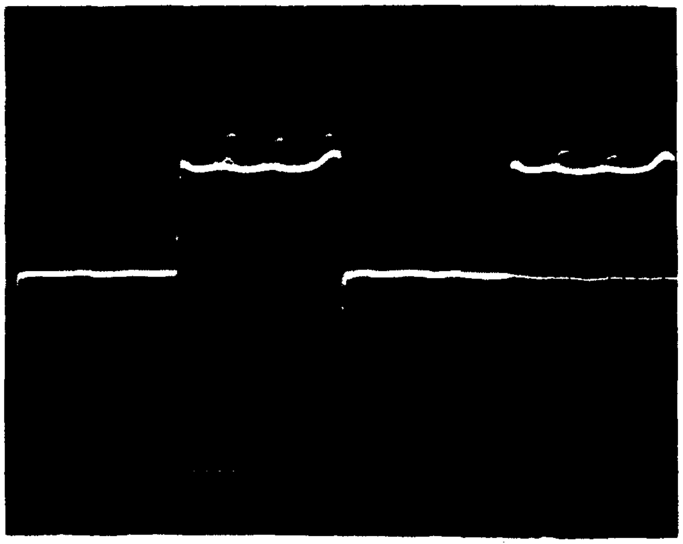
**Figure 6: Spillover Countermeasure Circuit**

# IV. EXPERIMENTAL RESULTS

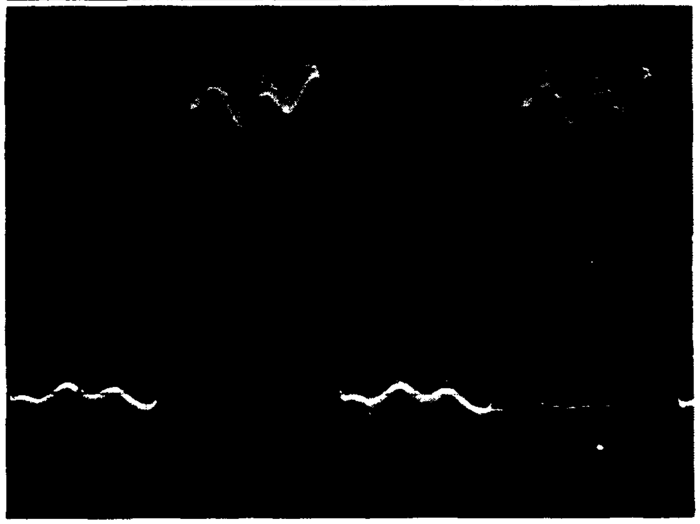
This chapter presents measured data from the redesigned transmit and receive circuits.

## A. TRANSMITTER

As shown in Figure 7, the transmit transducer output voltage increases from about + 4 volts to10 volts for the circuit in Figure 5 with both frequencies at 40 kHz (cycle is 25 microsecond).



(a) Supply voltage = + 5 volts (x: 1 msec./div y: 20 mV./div)



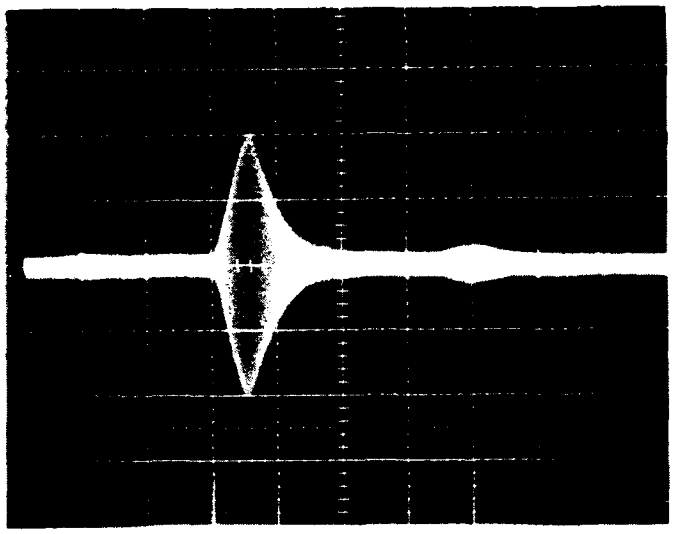
(b) Supply voltage = +12 volts (x: 1 msec./div y: 50 mV./div)

Figure 7: Transmit Transducer Output Voltage

## B. RECEIVER

### 1. Effect of Change in Transducer Output

When transmit transducer output voltage is changed from 4 volts to 10 volts, the receive transducer output voltage is increased as shown Figure 8.


**(a) Supply voltage = + 5 volts (x: 1 msec./div y: 20 mV./div)**

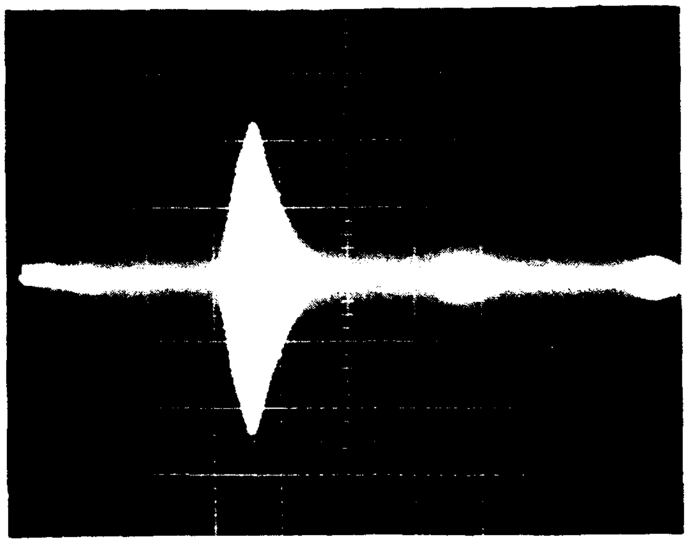

**(b) Supply voltage = +12 volts (x: 1 msec./div y: 50 mV./div)**

**Figure 8: Receive Transducer Output Voltage**

## 2. Effect of Spillover

After increasing the transmit transducer output voltage to 10 volts, the value of spillover increased. Figures 9 and 10 show a sonar driver board circuit and output voltage at points G and H on the circuit without spillover countermeasure circuit. The presence of this spillover prevents a correct range reading for the wall.

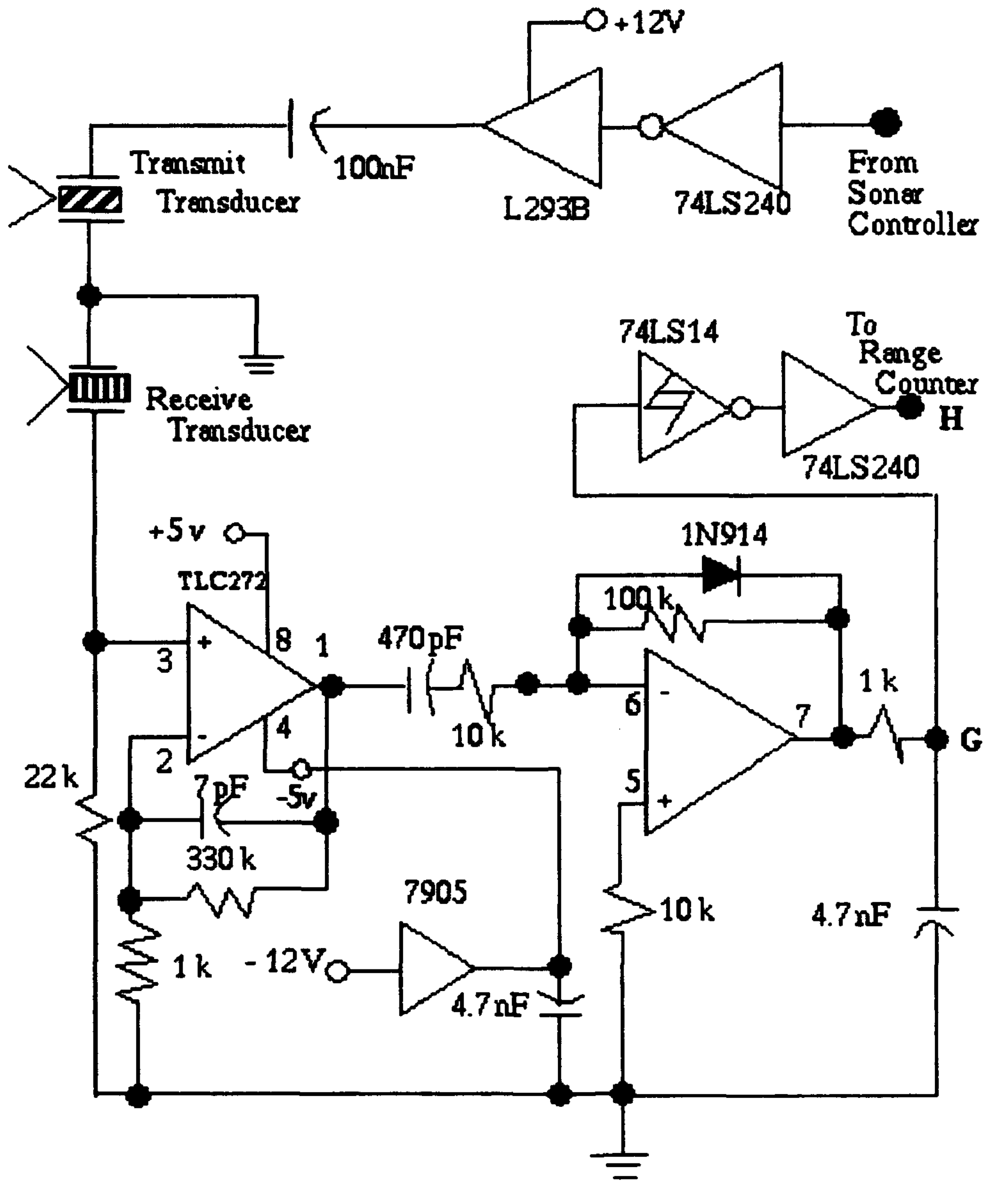
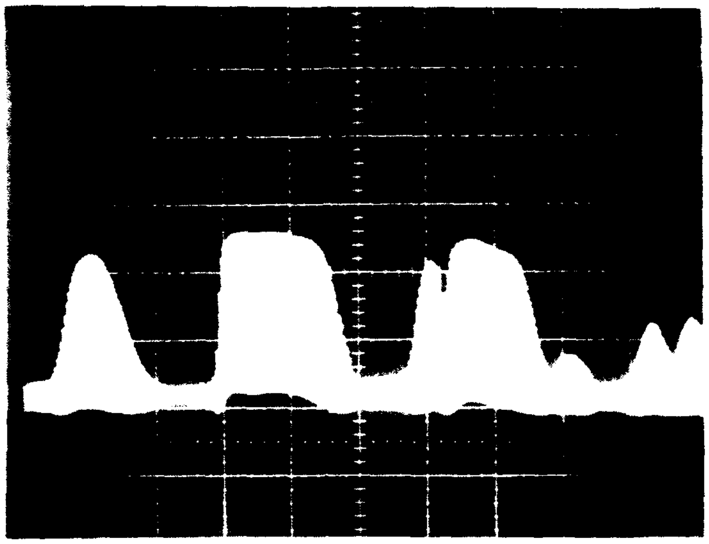
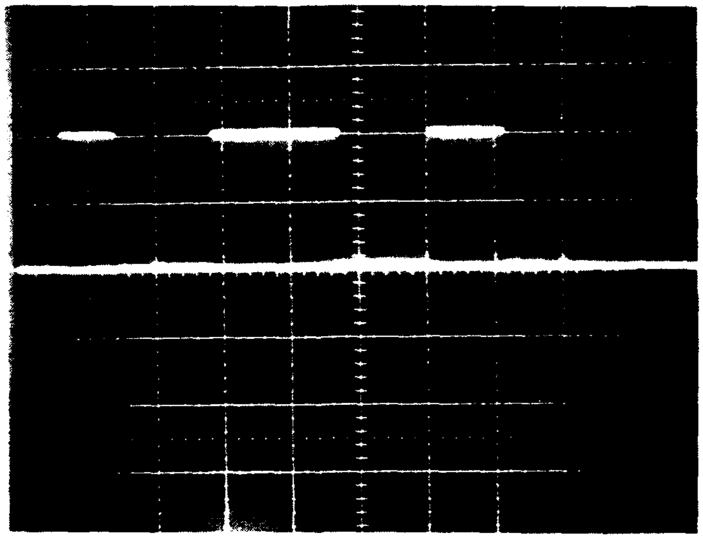


Figure 9: Sonar Driver Board (Spillover)



(a) Point G (x: 1 msec. /div y: 1 V./div)



(b) Point H (x: 1 msec./div y: 2 V./div)

**Figure 10: Output Voltage (Spillover)**

The new circuit with the spillover countermeasure incorporated (capacitance-resistance circuit) is shown in Figure 11. The output voltage at points G and H were measured and are shown in Figure 12.

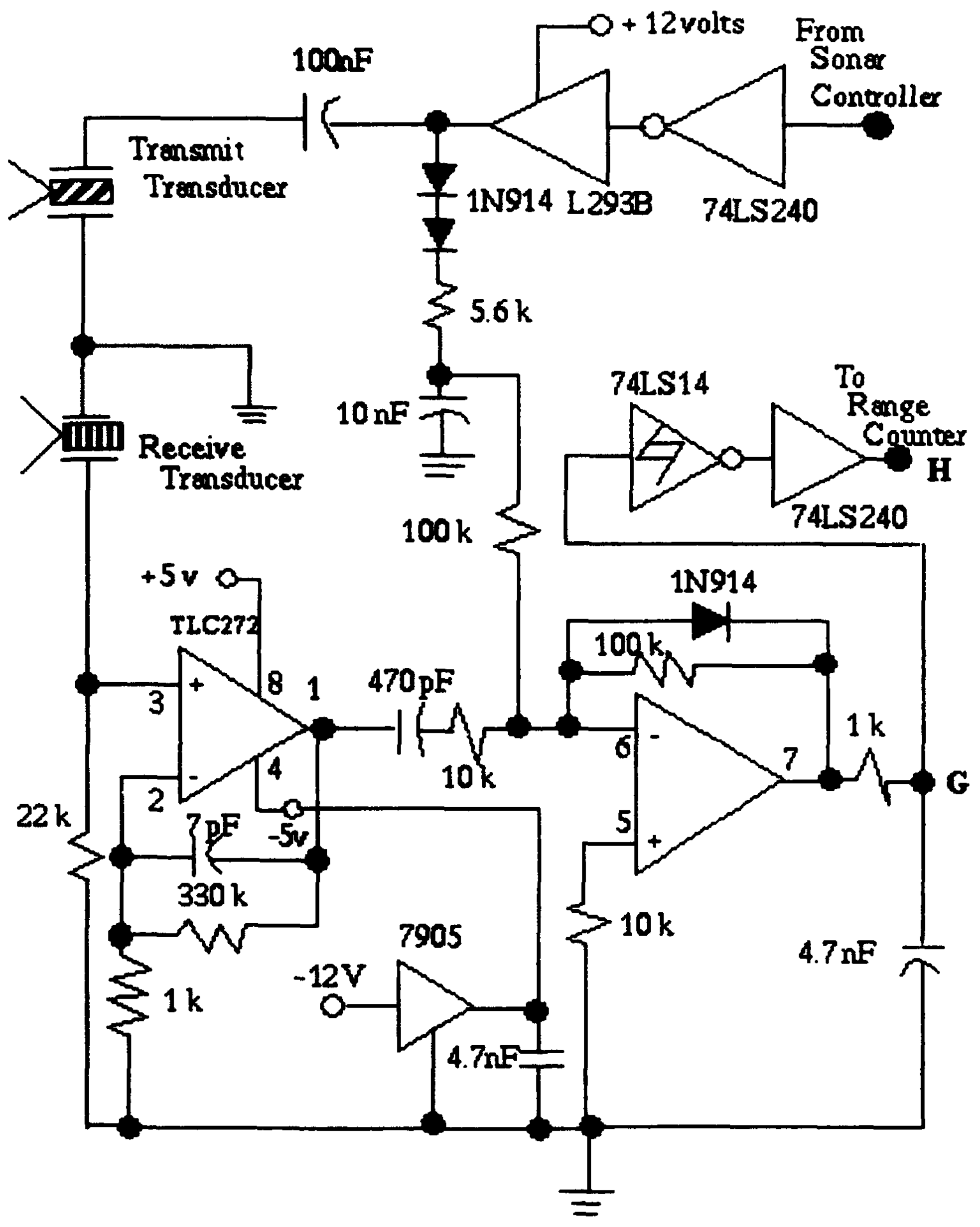
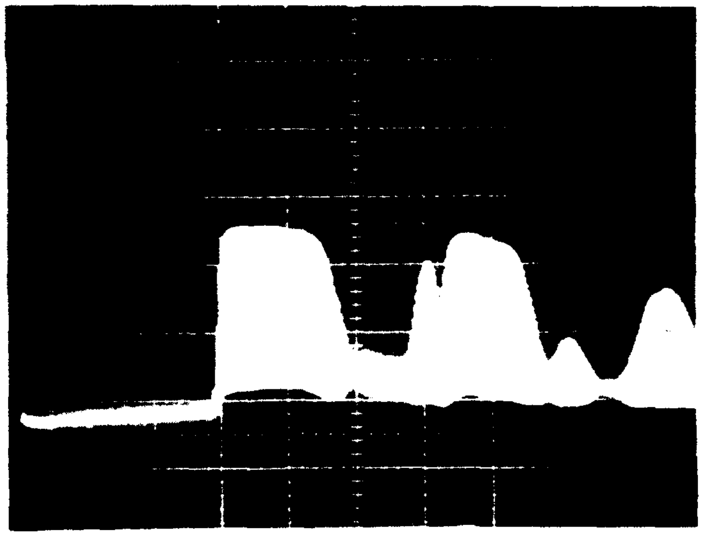
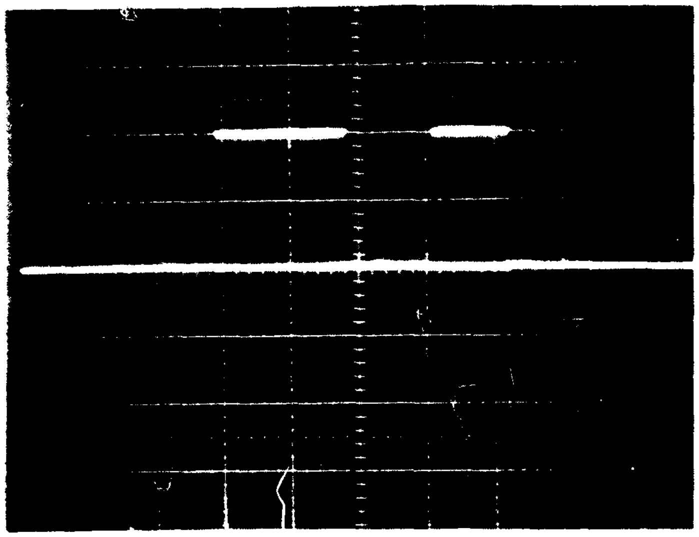


Figure 11: Sonar Driver Board (Spillover Countermeasure)

The photos show that the spillover output from 0.5 milliseconds to 3.0 milliseconds after transmit transducer output pulse disappears. Consequently, the correct range from the wall is obtained.



(a) Point G (x: 1 msec./div y: 1 V./div)



(b) Point H (x: 1 msec./div y: 2 V./div)

Figure 12: Output Voltage (Spillover Countermeasure)

## 3. Frequency Response

Figure 13 shows the test circuit for the frequency response. The output was measured while sweeping the input frequency from 8 Hz to 120,800 Hz, with the supply voltage of the sine wave constant with 200 millivolts peak-to-peak.

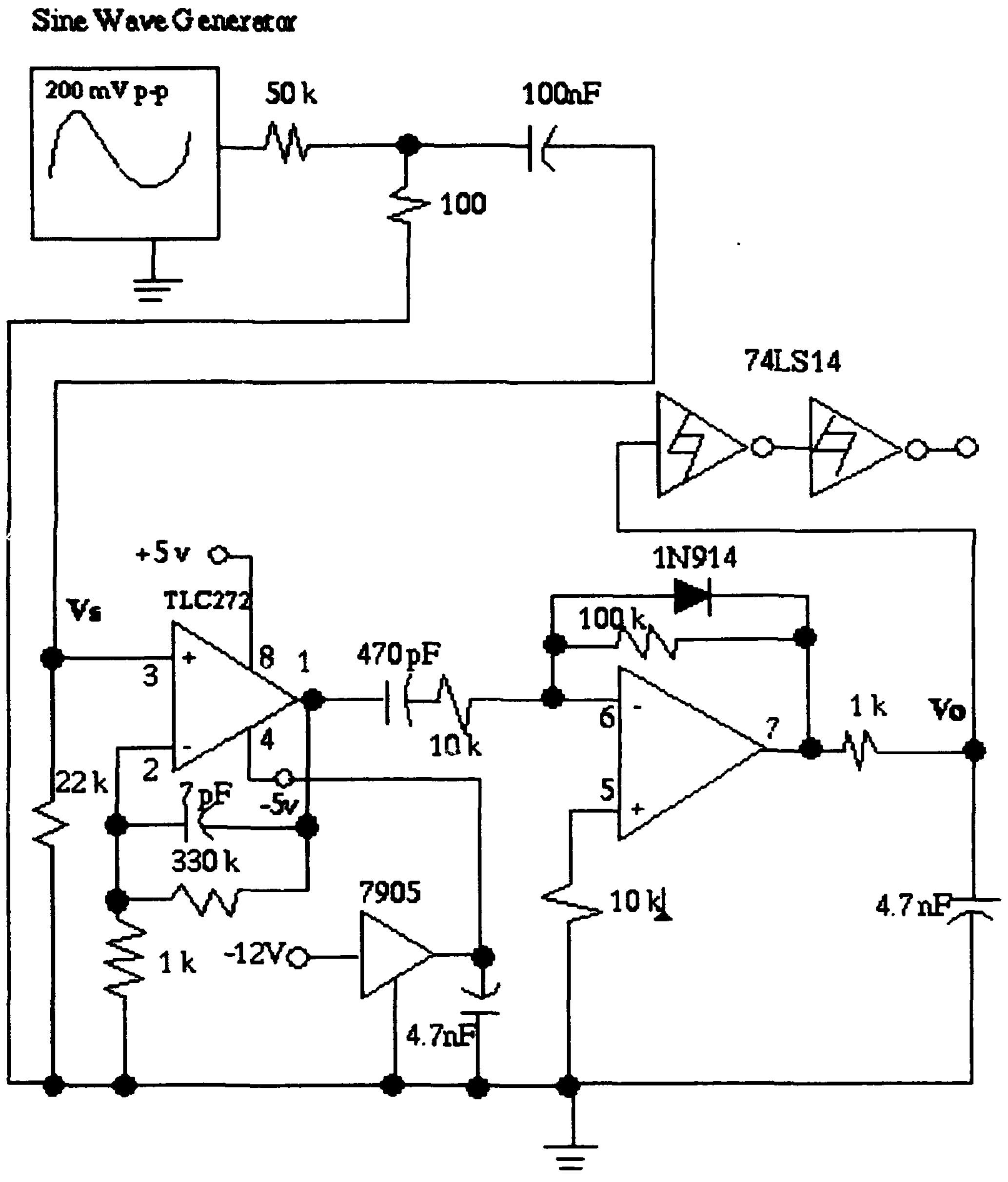


Figure 13: Test Circuit of Frequency Response

This measurement provides an indication of the noise immunity of this circuit. The result is shown in Figure 14. According to this result, a frequency of 40 kHz, which transmit/receive transducer uses, is close to the maximum frequency response.

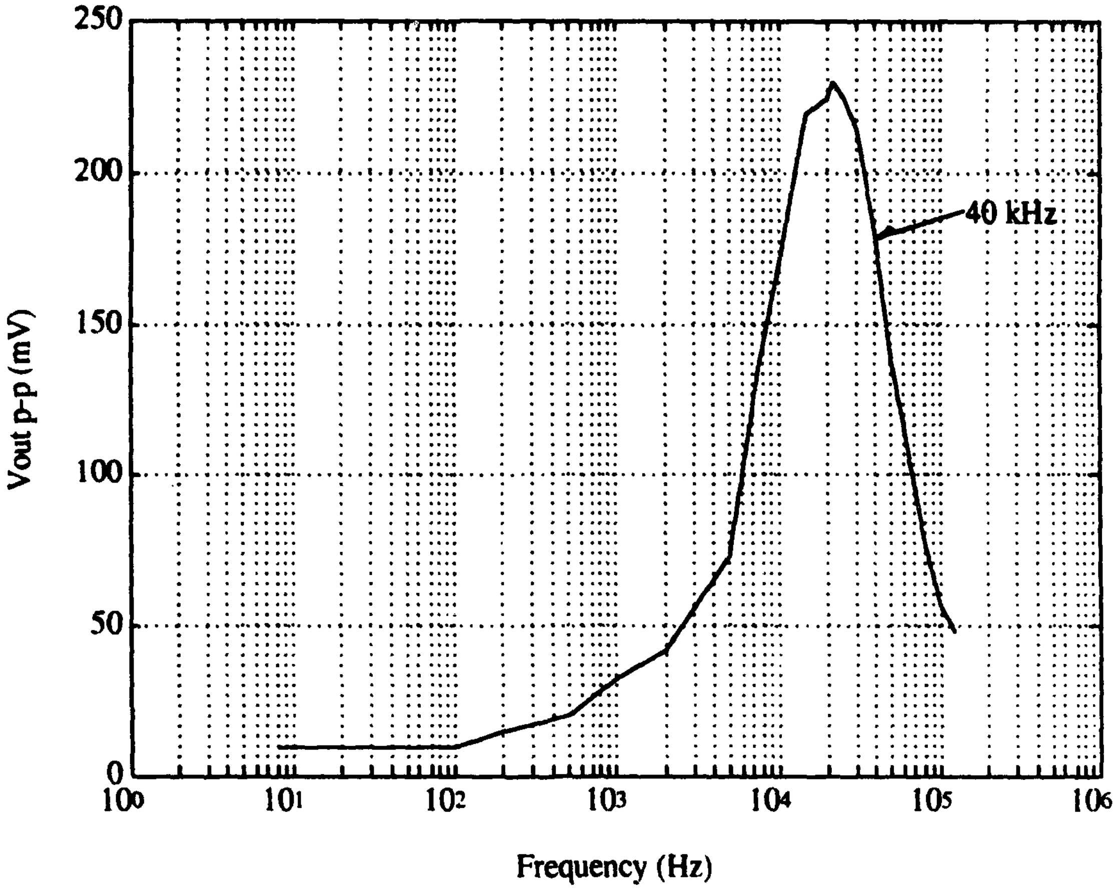


**Figure 14: Frequency Response**

### 4. Gain Measurement

The gain of the modified circuit was measured by using the test circuit shown as Figure 16, where the frequency of the square wave was 40 kHz. From the measurement, the 1st stage gain of operational amplifier and 2nd stage gain are as follows:

$$\text{1st stage gain} = 20\log(V1/Vin) = 20\log(20/0.2) = 40 \text{ dB} \quad (5.1)$$

$$\text{2nd stage gain} = 20\log(V2/V1) = 20\log(88/20) = 12.87 \text{ dB}. \quad (5.2)$$

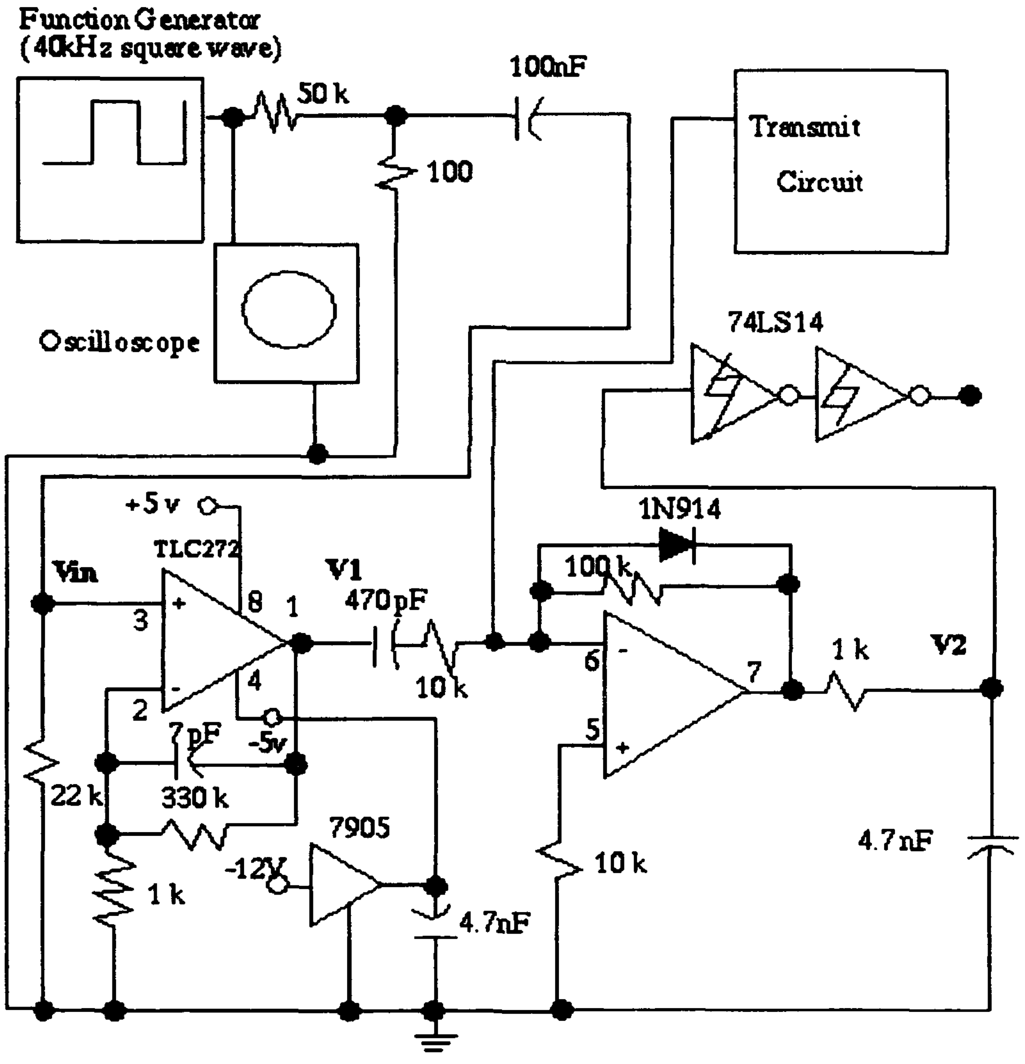


Figure 16: Test Circuit for Gain Measurement

## 5. Amplifier Sensitivity

The input voltage required for detection versus time from transmit pulse was measured in order to get the shape of the threshold envelop. Figure 16 shows the test circuit for this measurement. The result is shown in Figure 17. It shows a monotonic decrease in voltage with time after the transmit pulse is initiated, and becomes constant after about 6 milliseconds.

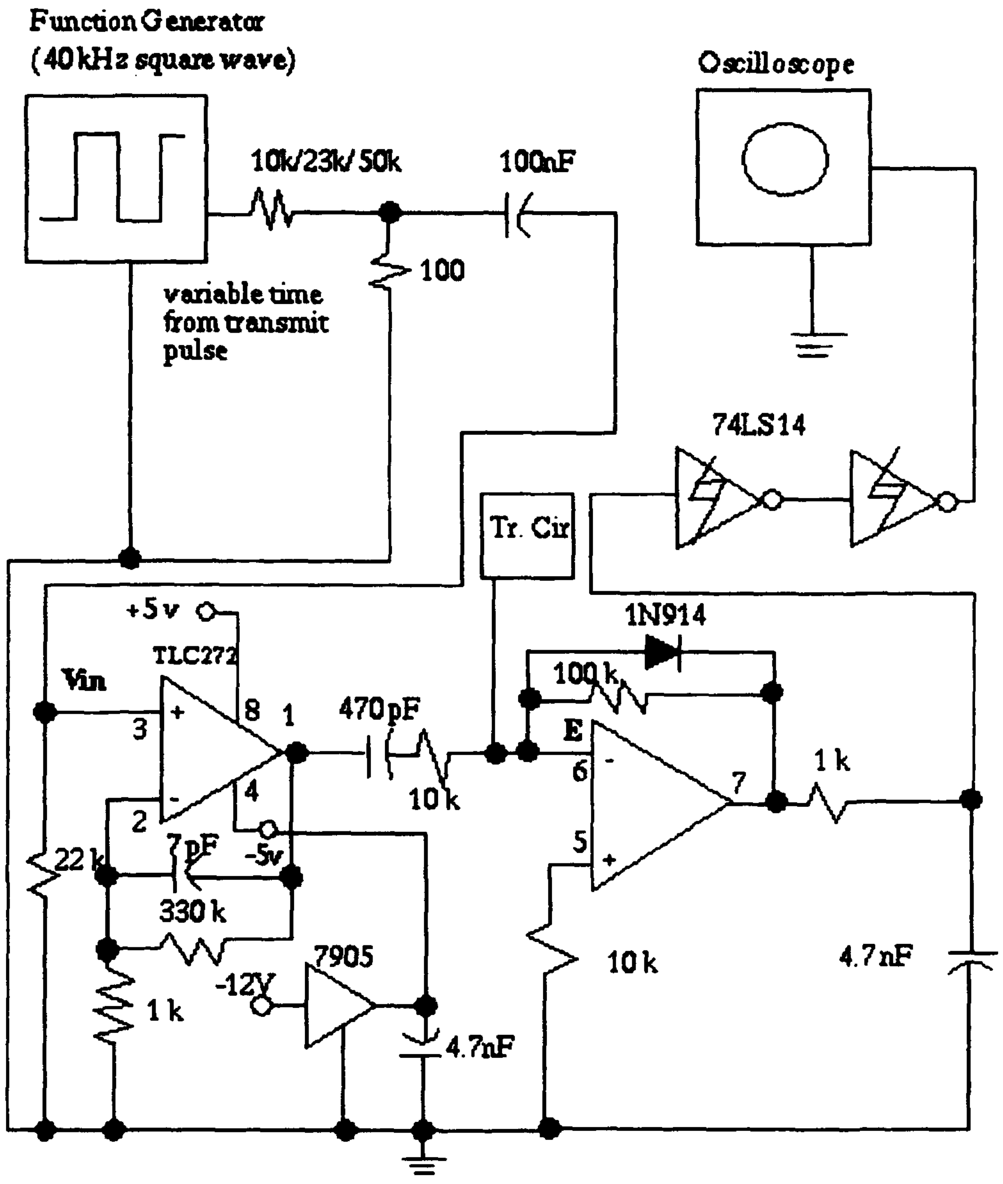


Figure 16: Test Circuit for Amplifier Sensitivity

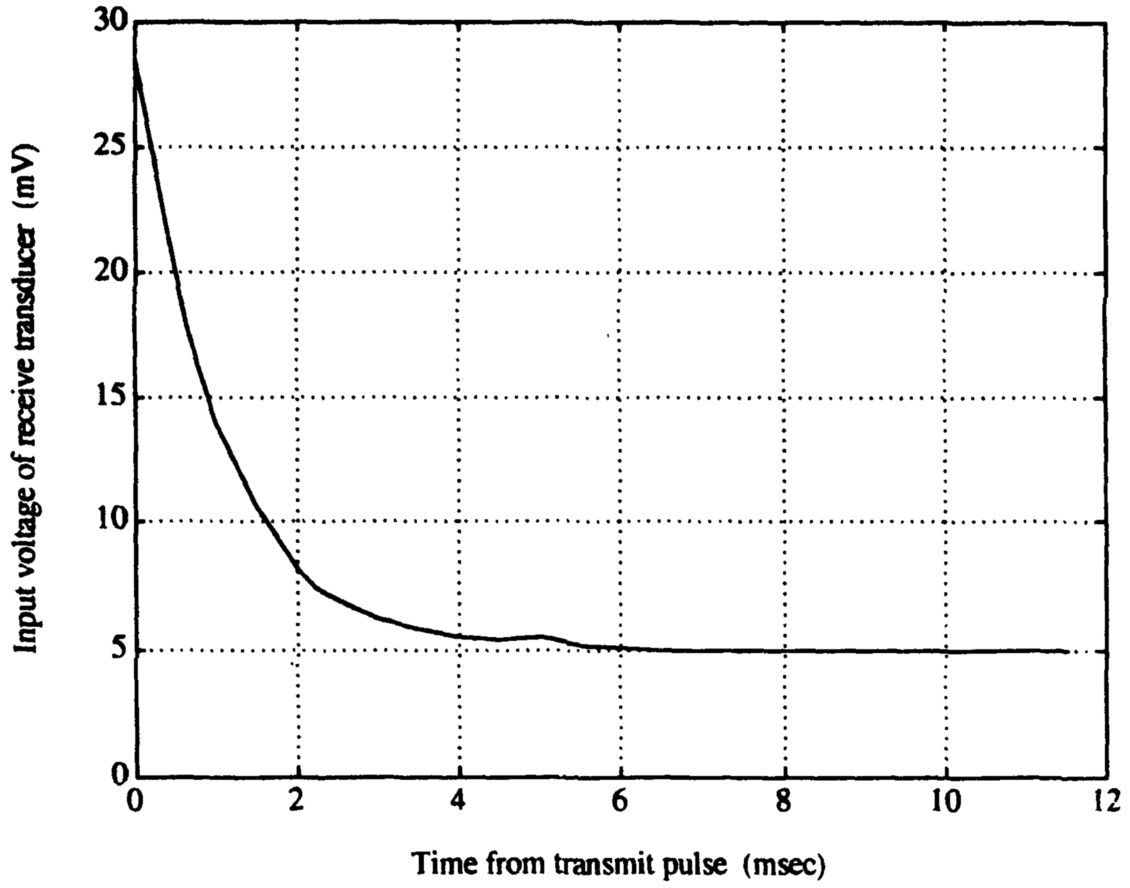


**Figure 17: Shape of Threshold**

## 6. Sensitivity Dependency on Distance

The sensitivity of the range measurement depends on the gain of the modified circuit. Figure 18 shows the gain measurement configuration. The gain as a function of distance is shown in Figure 19 for the new circuit.

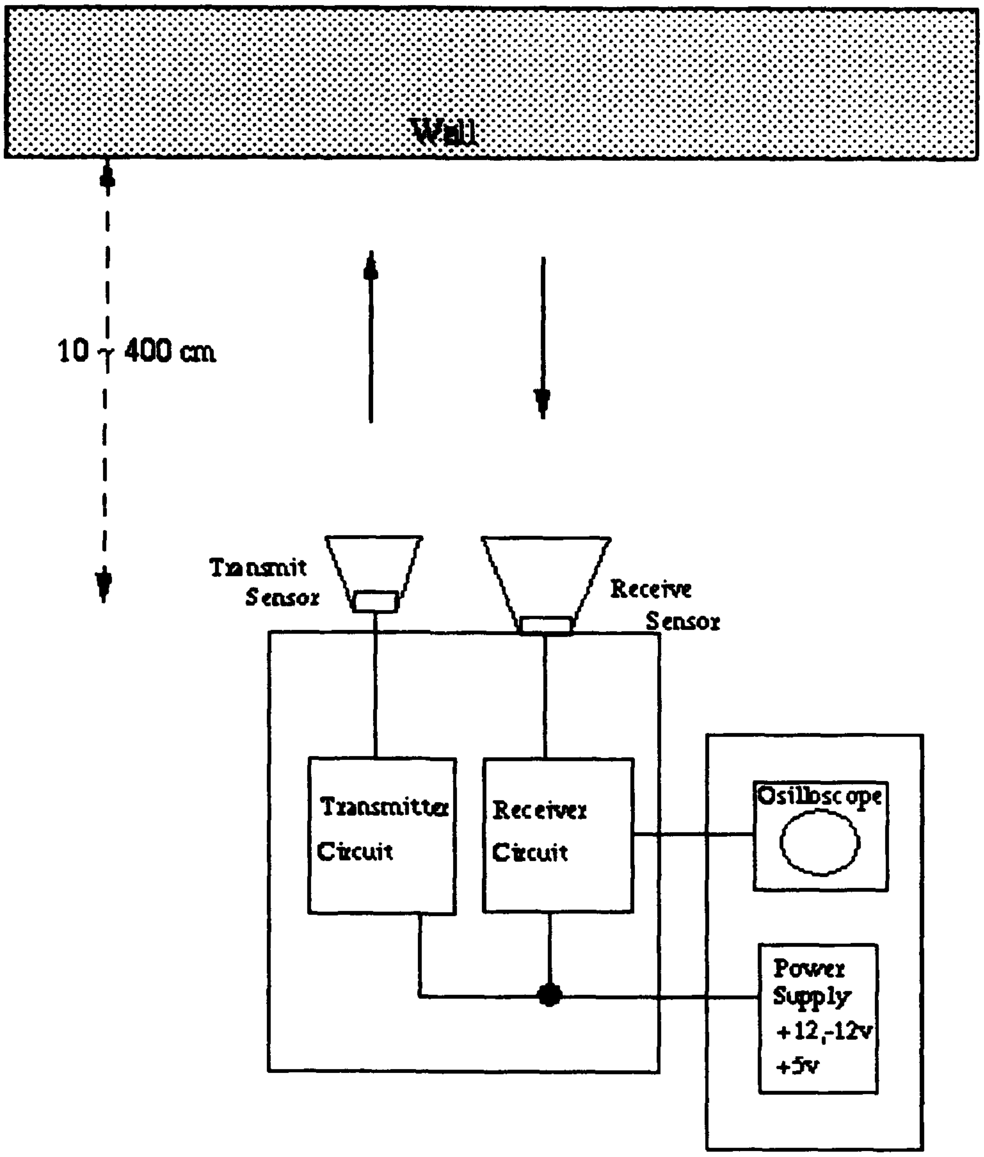


Figure 18: Test Method for Gain

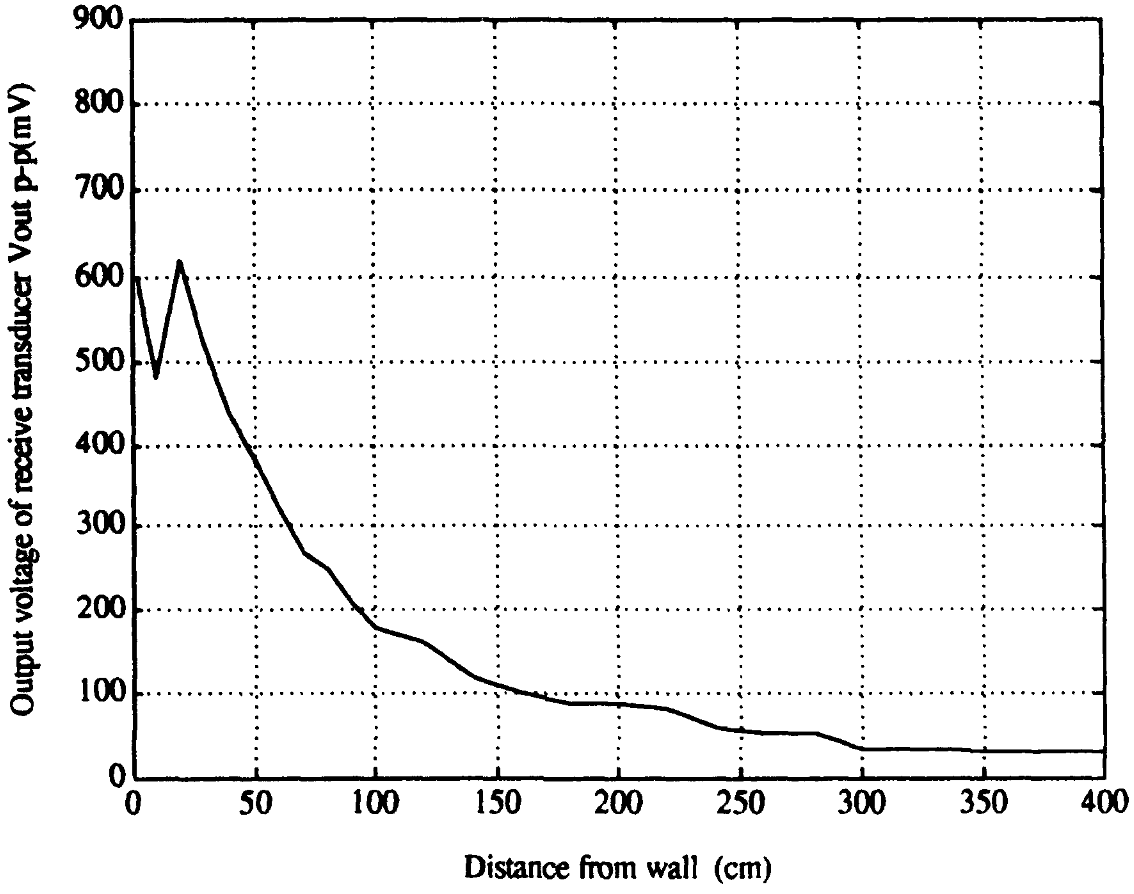


**Figure 19: Output Voltage of Receive Transducer (I)**

In Figure 20, gain data for the old circuit with 5 and 12 volt sources are shown along with the data for the new circuit. Both of the 12 volt cases have comparable performance, except at ranges less than approximately 25 centimeters.

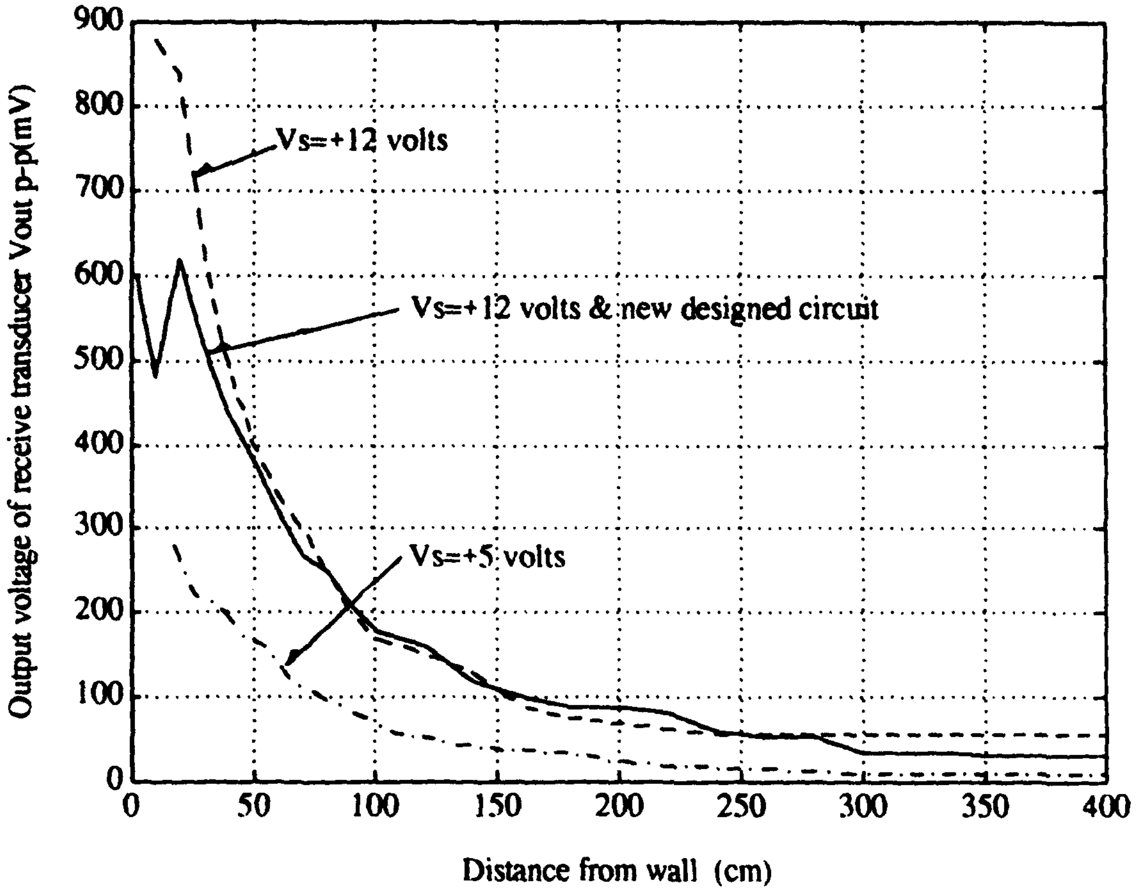
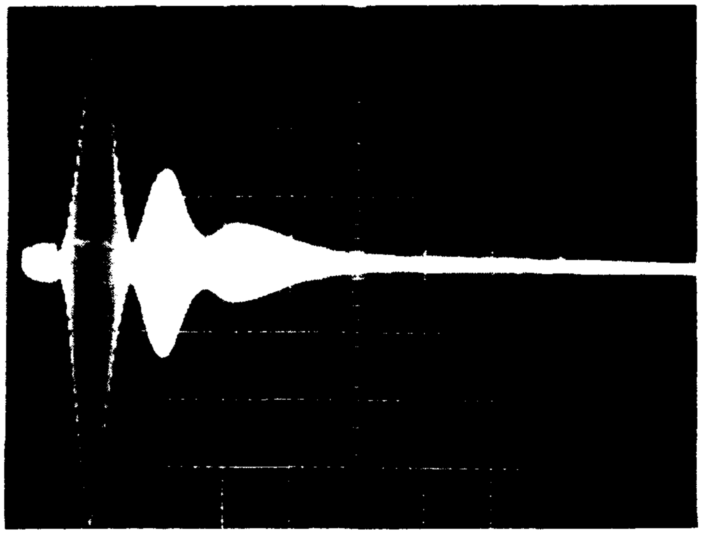


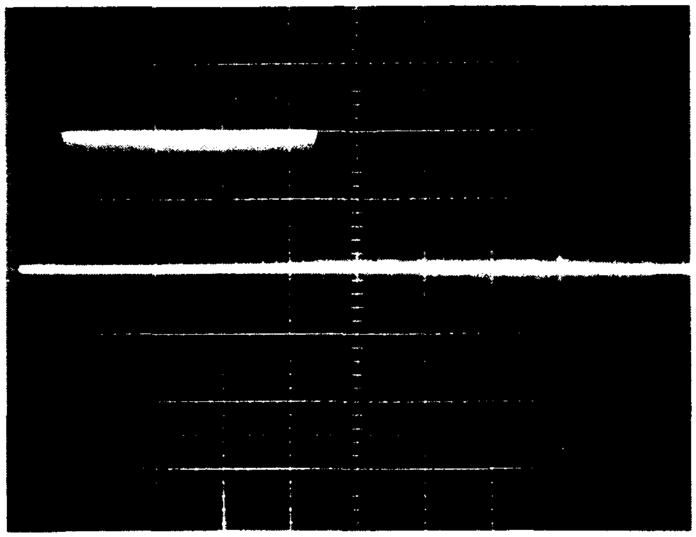
Figure 20: Output Voltage of Receive Transducer (II)

## 7. Minimum Range

With the test configuration shown in Figure 18, the minimum range was found. Figure 21 shows the output voltage of receive transducer and the output voltage from Schmitt T.igger (74LS14) chip.



(a) Receiver Transducer (x: 1 msec./div y: 50 mV./div)



(b) Schmitt Trigger (x: 1 msec./div y: 2 V./div)

Figure 21: Output Voltage for Minimum Range

According to this result, the minimum range lies approximately 2.0 centimeters outside the periphery of the robot

$$range = (34000\ centimeter/second) \times (0.6\ millisecond)/2 = 10\ centimeters \qquad (5.3)$$

Therefore, by using the modified circuit, an improvement in the minimum range was achieved; from 17 centimeters to 10 centimeters.

## 8. Noise Measurement

The receive transducer signal was measured while the robot was in operation (Appendix). There was significant noise at a frequency of 7.9 kHz (magnitude of 20 millivolts) caused by the driving motor with pulse width modulation. Figure 22 shows the output voltage of the receive transducer.

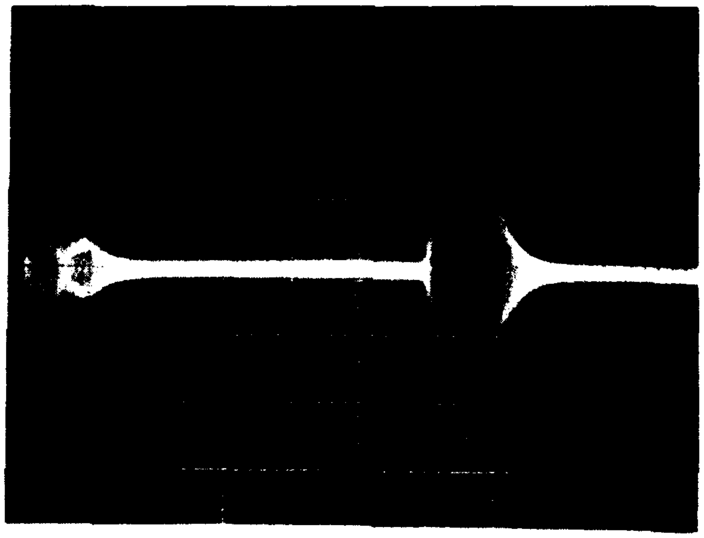


**Figure 22: Noise at Receive Transducer**

**(x: 1 msec./div y: 20 mV./div)**

Figure 23 shows output voltage of the pre-stage on Schmitt Trigger Inverter. This shows a period is 126.6 microseconds (1/7900 = 126.6) and a magnitude of 0.14 volts. Therefore, the noise is small enough so that a Schmitt Trigger Inverter can be used (minimum trigger voltage = 1.4 volts).

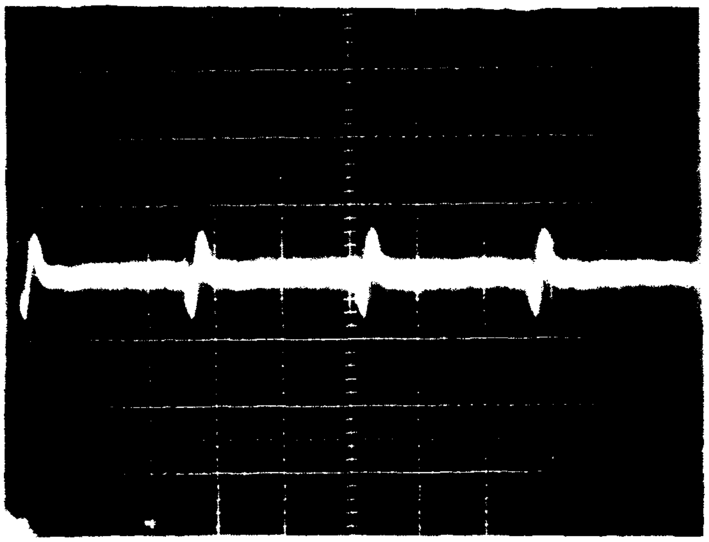


**Figure 23: Noise at Pre-Stage of Schmitt Trigger Inverter (I)**
**(x:50 usec./div y: 0.1 V./div.)**

Figure 24 shows output voltage of pre-stage on Schmitt Trigger Inverter after the circuit was installed permanently on the driver board. This result indicates a noise level of 0.28 volts. Therefore, the modified receiver amplifier circuit was not affected by motor noise.

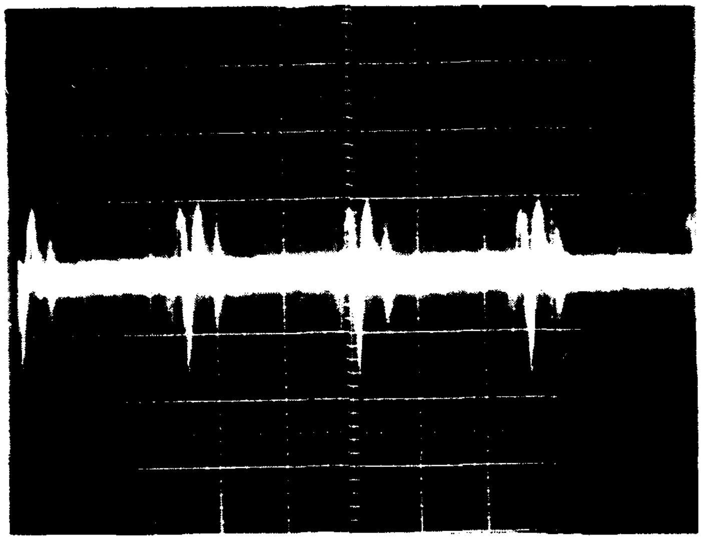


**Figure 24: Noise at Pre-Stage of Schmitt Trigger Inverter (II)**

**(x:50 usec./div y: 0.1 V./div.)**

# V. CONCLUSIONS

## A. RESULTS

The performance of the sonar hardware system on the autonomous mobile robot Yamabico-11 was improved. By changing the driving voltage of the transmit transducer from 5 volts to 12 volts, the receive gain was increased threefold. The sonar system is now capable of reliably detecting an obstacle at a distance of four meters. A disadvantage of increasing the voltage was an increase in spillover directly from the transmitter. The spillover interference was cut by the addition of capacitance-resistance circuits and negative voltage source for suppressing the gain. Also, the modified circuit had the added advantage of shortening the minimum range to 10 centimeters. The new design was fabricated and installed permanently. Tests on the installed circuit showed that it was not affected by noise.

## B. RECOMENDATIONS

The remaining sonar driver circuits should be upgraded, and testing should be done to ensure that the sonar parameters are optimized. Finally, the effectiveness of the sonar in obstacle avoidance must be studied.

# APPENDIX

This appendix contains the C code for sonar testing and the user file that were used in this thesis.

## APPENDIX A. SONAR TEST CODE

```
# sonartest3.s
#
# This program adds the "H", "V", and "G" commands to the
sonar test program.
#
# Typing an "H" puts the program in HUSH mode.  The program
continually ranges
# without calculating any values or printing anything to
the screen.  This
# allows generating totally evenly spaced pings.  Typing
"V" restores the
# program to VERBOSE mode, where it calculates the average,
quality, and
# hits, and prints them to the screen.
#
# Typing a "G" results in the program printing a "GAP=?" to
the screen.
# A DECIMAL value terminated by a <CR> may be typed.  This
value will be
# the delay between the end of the last range cycle and the
beginning of the
# next range cycle.  Values are in milliseconds.  Typing
"0" or just a <CR>
# sets this delay to zero.  This delay applies when the
program is in HUSH
# or verbose mode.
#
# System range test program.
# Any ONE sonar may be selected by typing its number.  Use
"A" for sonar
# 10 and "B" for sonar 11.  Use "C", "D", "E", and "F" for
the four test
```

```
# values.  Type "Q" to exit the program.
#
# The constant "samples" is the number of range
measurements that will
# be taken and averaged for the final range
determination.  This average
# value, or nominal range will be printed to the control
terminal.
#
# Each of the range values that is not an overrange will be
added to the
# others, then divided by the number of non-overrange
values.  This is the
# nominal value.  Then the number of range values that are
within plus or minus
# "qwidth" millimeters of the nominal value are
determined.  The percentage of
# values within # this spread is the quality.  Finally, the
percent of
# non-overrange values to overrange values is the hit rate.
#
#
# Assemble and link the program with the following two
command lines:
#
# as -o sonartest2.o sonartest2.s
# ld -s -T 304000 -e beg -o sonartest2.out sonartest2.o
#
# Then log into the yamabico account at the robot-terminal
shared line.
# Type "cs" (this sets up aliases and cd's to the "mml"
subdirectory).  Copy
# the "sonartest2.out" program into the "mml"
subdirectory.  Turn on the robot,
# and switch the serial line from the terminal to the
robot.  At the robot
# console type:
# lo = dload sonartest2.out
#
# Ignore the checksum error messages and type:
#
# g 304000
#
```

```
# to start the program.


# Samples is the number of sonar samples in each test cycle.

samples    =          32

# Qwidth is the number of millimeters plus or minus the
nominal value that
# a data value can be and be counted as a quality value.

qwidth     =          5                                | +-5 mm
is 1 cm

# Ascii codes

aspace  =        0x20                      | ascii space code
azero   =        0x30           | ascii "0" code
anine      =          0x39                             | ascii
"9" code
quest      =          0x3f                             | ascii
"?" code
ana        =          0x41                             | ascii
"A" code
anf        =          0x46                             | ascii
"F" code
agee       =          0x47                             | ascii
"G" code
anh        =          0x48                             | ascii
"H" code
aque       =          0x51                             | ascii
"Q" code
avee       =          0x56                             | ascii
"V" code

# 1 ms timing loop count value.  Adjust if timing loop
modified.

tlcnt      =          0x9                              | timing
loops for 1 ms delay

# Sonar board status register.  Bit 7 (0x80 hex) indicates
busy.  Bits 4 and
```

```
# 3 encode the current sensor group

statreg =       0xffff83f9                      | sonar board
status register address
bsyb    =       0x80                            | busy bit
sgmask  =       0x18                            | sonar group mask

# Sonar board data registers

s0data  =       0xffff83f0                      | sonar board data
registers
s1data  =       0xffff83f2
s2data  =       0xffff83f4
s3data  =       0xffff83f6                      | sensor #3 data
register address

# Sonar board overrange indicator value

overrng   =         0x8000                                |
overrange value


        .globl  beg                             | let loader access
first instruction
        .text

# Write out an introduction line

beg:    movl    #stmesg, a0                     | get start address
of message
        movl    #stmese, a1                     | get end address
of message + 1
          jsr prline                            | print to console

# Initialize variables

          movl      #0, d0                                | get 0
          movl      d0, hushf                   | clear hush mode
flag
          movl      d0, gapsiz                  | set gap to 0

# Initialize next sample cycle
```

```
wl2:     movl     #samples, pingct| initialize ping
counter
         movl     #sdata, saddr      | initialize
storage counter

# Wait for next sonar data or key press

wl1:     movl     gapsiz, d0         | get gap value
         movl     d0, gapval         | reset delay
counter
        trap    #15                  | get tty input
status
        .word   0x0001               | .instat function
        jeq     sdl                  | no key pressed,
continue

# Key pressed

wl6:    subql   #2, sp               | make space on
stack for character
        trap    #15                  | get character
from input
        .word   0x0000               | .inchr function
        movb    sp@+, d0             | put char in d0,
fix stack
        andl    #0x7f, d0            | mask any parity
bit
        jsrselson               | select sonar from
pressed key
         jmp      wl2                          | restart
data collection

# Check if a sonar group is ready

sdl:    movb    statreg:l, dl        | get sonar status
        andb    #bsyb, dl            | see if busy flag
set
        jne     wl1                  | if still busy loop

# Gap after sonar completes

wl5:     movl     gapval, d0         | get delay count
         andl     #0xfffffff, d0     | check for zero
```

```
            beq       wl4                                    | if
delay expired continue
            subl      #1, d0                                 |
decrement delay count
            movl      d0, gapval              | store new gap
value
            trap    #15                         | get tty input
status
          .word   0x0001                      | .instat function
          jeq     wl5                         | no key pressed,
continue
            jmp       wl6                                    | key
pressed, so handle

# Check for HUSH mode after sonar completes

wl4:        movb      hushf, d1               | get hush mode flag
            andb      #0xff, d1               | check if hush
flag set
            jeq       wl3                                    | not
hush mode, process data

# Hush mode, restart sonar and continue looping

            movw      s3data:1, d0            | restart sonar
            jmp       wl1                                    |
continue looping

# Verbose mode, store data and decrement ping count

wl3:        movl      saddr, a0               | get data table
pointer
            movl      sdreg, a1               | get sonar
register address
            movw      a1@, d0                                | get
sonar data
            movw      d0, a0@+                | store in data
array
            movw      s3data:1, d0            | restart sonar
            movl      a0, saddr               | update data
pointer
            subl      #1, pingct              | decrement ping
count
```

```
          jne      wl1

# Compute results

          jsr      cmpav                                | compute
average range, and hits
          jsr      cmpqu                                | compute
Q value
          jsr      cnvqh                                | make Q
and hits percent values
          jsr      prdata                               | print
data to console

          jmp      wl2                                  | begin
next cycle

# Return to on board debug monitor

leave:  movb    #0, d0                | turn off sonars
        movb    d0, statreg:1
        movw    s3data:1, d0          | flush any old data
        trap    #15                   | return to debug
        .word   0x0063                | .return function

# Select Sonar.  Call with key press data in d0.  This
routine looks up
# the corresponding sonar and sets the sonar number in the
data message,
# sets the  correct sonar data address in "sdreg", and
writes the correct
# group select to the sonar command register.


selson:   cmpb     #quest, d0           | is it letter or
number ?
          bgt      selno1                               | go
handle letter
          cmpb     #azero, d0           | is it less than
"0"
          blt      selex                                | if so
bad input so exit
          cmpb     #anine, d0           | is it greater
than "9"
```

```
          bgt       selex                             | if so
bad input so exit
          subb      #azero, d0              | normalize ascii
to digit
          jmp       seltbl                            | and
load values from table

selno1:   andb      #0x5f, d0               | make lower case
upper case
          cmpb      #aque, d0               | is input a "Q"
          beq       leave                             | if so
exit
          cmpb      #anh, d0                | is input "H"
          bne       selno2                            | skip
next if not
          movb      #0xff, hushf            | set hush flag on
          movl      #husmsg, a0             | get hush messages
start
          movl      #husmse, a1             | get hush message
end
          jsr       prline                            | print
hush message
          rts

selno2:   cmpb      #avee, d0               | is input "V"
          bne       selno3                            | skip
next if not
          movb      0x0, hushf              | set hush flag off
          rts

selno3:   cmpb      #agee, d0               | is input "G"
          beq       setgap                            | go to
set gap routine

          cmpb      #ana, d0                | is it less than
"A"
          blt       selex                             | if so
bad input so exit
          cmpb      #anf, d0                | is it greater
than "F"
          bgt       selex                             | if so
bad input so exit
          subb      #0x37, d0               | normalize ascii
```

```
to digit

seltbl:   roll      #3, d0                           |
multiply digit by 8 for offset
          movl      #sptabl, a0            | get table start
address
          addl      d0, a0                           | form
data table entry address
          movl      a0@+, sdreg            | load data
register pointer
          movw      a0@+, dtamsg           | load sensor
number in data string
          movb      a0@, statreg:1         | load group to
sonar command register
selex:    rts


# Setgap routine.  Print the message "DELAY=? ", then read
a DECIMAL value
# up to 9999.  The decimal value is delay in milliseconds
between the end
# of one ranging cycle and the beginning of the next.  Set
the Delay value.

setgap:   movl      #gapmsg, a0            | get start of
message
          movl      #gapmse, a1            | get message end
          jsr       prstng                           | print
message
          jsr       rdline                           | read
delay value into buffer
          jsr       cnvtsn                           | convert
to value in d0
          bne       seterr                           | if not
zero return error
          mulu      #tlcnt, d0             | multiply 1 ms
loop count by ms needed
          movl      d0, gapsiz             | initialize gap
count
          rts
seterr:   movl      #derrms, a0            | get error message
          movl      #derrme, a1            | get message end
          jsr       prline                           | print
```

```
error message
          jmp       setgap                        | and try
again


# Convert text string to number.  Call with the address of
the text string
# count byte the a0 register.  Return with the converted
value in the d0
# register.  If the number of characters is zero return with
# a value of 0.  If the string converts correctly return
with zero flag set,
# otherwise return with zero flag clear.

cnvtsn: movl#0, d0                      | setup d0 as total
accumulator
          movb      a0@+, d1            | get count byte
into d1
          cmpb      #4, d1                        | is it
greater than 4?
          bgt       cnvts3                        | if so
error
cnvts1:   andb      #0xff, d1           | is d1 zero
          bne       cnvts2                        | not
zero, continue
cnvts3:   rts                                     | else
done

cnvts2:   movb      a0@+, d2            | get character
          jsr       cnvdig
          blt       cnvts3                        | if
minus, error
          mulu      #10, d0                       |
multiply previous total by 10
          addw      d2, d0                        | add in
new data
          subb      #1, d1                        | reduce
char count
          jmp       cnvts1                        | continue

# Convert digit.  Call with the ascii code for a decimal
number between
# 0 and 9 in the d2 register.  Return with the value in the
```

```
d2 register.
# If the ascii code is outside the correct range return
with minus flag
# set.

cnvdig:   andl     #0x7f, d2              | mask unneeded bits
          subb     #azero, d2             | normalize ascii
to number
          blt      cnvdil                           | if
ascii below zero code error
          movb     #9, d3                           | check
for above range
          cmpb     d2, d3
cnvdil:   rts

# Compute Average Value.  This routine computes the average
value of the
# data in the "sdata" array.  This average value is placed
in "nomval".
# Note that ONLY non-overrange values are computed.  The
number of
# non-overrange values in the table is saved in "hitval".
If ALL values
# are overrange "nomval" is set to 0x8000 and "hitval"
equals 0.

cmpav:    movl     #sdata, a0             | use a0 to point
to values
          movl     #0, d0                           |
initialize d0 to accumulate sum
          movl     d0, d1                           |
initialize d1 to accumulate hits
          movl     #samples, d2           | initialize d2 as
test counter
cmpl1:    movw     a0@+, d3               | get value into d3
          cmpw     #overrng, d3           | test for overrange
          beq      cmpl2                            | if
overrange do not add or incr hits
          addl     d3, d0                           | add to
d0 total
          addl     #1, d1                           |
increment hit counter
cmpl2:    subl     #1, d2                           |
decrement loop count
          bne       cmpl1:

          movl      d1, hitval          | set hitval

          cmpl      #0, d1                        | check
for no hits
          bne       cmpl3                         | if hits
go and handle
          movl      #overrng, nomval| indicate all overrange
          rts                                     | finished

cmpl3:    divu      d1, d0                        | compute
average range
          andl      #0xffff, d0         | discard remainder
          movl      d0, nomval          | store in nomval
          rts                                     | finished

# Compute quality.  Call AFTER "compute average value"
routine has set
# "nomval" and "hitval".  This routine scans the data in
the "sdata" array.
# Every value within pl'  or minus "qwidth" millimeters of
the average
# value is counted as a quality value.

cmpqu:    movl      #sdata, a0          | use a0 to point
to values
          movl      nomval, d1          | get average value
          movl      #qwidth, d2         | get qwidth
          addl      d1, d2                        |
calculate high value in d2
          subl      #qwidth, d1         | calculate low
value in d1
          movl      #samples, d4        | put loop count in
d4
          movl      #0, d3                        | set d3
as compare counter

cmpql1:   movw      a0@+, d0            | get value
          cmpl      d2, d0                        | is
value > max hi
          bgt       cmpql2                        | if
```

```
value > max hi do not incr Q
          cmpl      d1, d0                        | is
value < min lo
          blt       cmpql2                        | if
value < min lo do not incr Q
          addl      #1, d3                        | quality
value, incr Q
cmpql2:   subl      #1, d4                        | decr
loop count
          bne       cmpql1                        | test
remaining values
          movl      d3, qval            | set qval value
          rts


# Convert Q and hits to percent.  This routine converts the
Q value to a
# percent of the total hits, and the hits value to a
percent of the total
# samples.  The values are then written back to the "qval"
and "hitval"
# locations.

cnvqh:    movl      #qval, a0           | get qval address
          movl      hitval, d1          | get hitval
          jsr       ctpcnt                        | make
qval percent of hits

          movl      #hitval, a0         | get hitval address
          movl      #samples, d1        | get samples
          jsr       ctpcnt                        | make
hitval percent of samples
          rts

# Convert to Percent.  This routine converts the ratio of
two quantities
# to a percent value between 0 to 100.  Call with the "a0"
register
# pointing to the numerator, and with d1 containing the
denominator.
# The percent of the two values will be written to the
location pointed
# to by "a0".
```

```
ctpcnt:   cmpl      #0, d1                                  | check
for zero denominator
          bne       ctpct1                                  | if not
zero continue
          movl      #0, a0@                                 | set
percent to zero
          rts
ctpct1:   movl      a0@, d0                                 | get
value to convert
          mulu      #100, d0                     | multiply by 100
          divu      d1, d0                                  | divide
by denominator
          andl      #0xffff, d0                  | mask remainder
          movl      d0, a0@                                 | store
converted value
          rts

# Convert Hex to decimal ascii.  Call with a hex value less
than or equal to
# 8191 decimal in the d0 register.  The a0 register is a
pointer to the
# memory location where the ascii equivalent will be
stored.  Leading zeros
# will be suppressed and replaced by spaces.  If the input
value is > 8191
# the word " OVER" will be printed instead of a number.


hexasc: movl    d0, d3                           | test input value
for > 8191
        andl    #0xffffe000, d3                  | and zero d3 if
valid
        jne     ovrrng                           | go to routine to
print " OVER"

# d3 is the leading zero suppress flag.  If it is zero then
we are still
# suppressing zeros.  It will be set to 0xFF when a non-
zero data value is
# encountered.

        movl    #3, d2                           | use d2 for the
```

```
divide loop count
        movl    #divsr, a1                   | use a1 to point
to divisors

# The divide loop divides d0 by 1000, then 100, then 10.
Each time the
# quotient is converted to ascii and written to the line
assembly area,
# and the remainder is placed in d0 to be divided in the
next loop.

dvloop: movl    a1@+, d1                     | put divisor in to
d1, inc a1
        divu    d1, d0                       | divide by power
of 10
        movw    d0, d1                       | move quotient to
d1
        andl    #0xf, d1                     | mask out all but
low byte
        bne     dvlop1                       | non-zero so skip
lead zero test

# Quotient is zero.  Check lead zero flag (d3 = 0) to see
if zero should be
# suppressed by substituting a space.

        andb    d3, d3                       | set zero flag if
d3 still = 0
        jne     dvlop1                       | zero suppress off
- go print zero
        movb    #aspace, d1                  | use space to
suppress leading zeros
        bra     dvlop2                       | go print space

# Store the ascii value in line assy area pointed to by a0

dvlop1: movb    #0xff, d3                    | turn off lead
zero suppress
        addb    #0x30, d1                    | form an ascii
number
dvlop2: movb    d1, a0@+                     | store in line
assy area
```

```
# Get remainder ready for division by the next power of 10

        swap    d0                          | move remainder to
lower word
        andl    #0xffff, d0                 | mask out old
quotient part

# Test for end of divide loop

        subb    #1, d2                      | decrement loop
count
        jne     dvloop                      | loop if not done

# End of loop.  Last remainder is last digit.  Convert to
ascii and store
# in line assy area.

        addb    #azero, d0                  | form ascii digit
        movb    d0, a0@

        rts                                 | return

# Handle overrange condition

ovrrng: movl    #5, d2                      | used d2 as loop
counter
        movl    #ormsg, a1                  | get address of
message

orrnl:  movb    a1@+, a0@+                  | move bytes
        subb    #1, d2                      | decrement count
        bne     orrnl                       | loop till copy
complete
        rts

# Powers of ten used to convert from hex to ascii

        .even

divsr:  .long   1000                        | divide by 1000
        .long   100                         | then by 100
        .long   10                          | then by 10
```

```
# Overrange message

ormsg:  .ascii  "OVER"

# Assemble and print data to console.  The nomval, Q, and
hits values are
# written to the data message assembly string, and then the
string is
# printed to the console.

prdata:   movl      #rdta, a0             | get data address
in assembly area
          movl      nomval, d0            | get range value
          jsr       hexasc                          | decimal
value to assembly area

          movl      #qdta, a0             | get qval address
in assembly area
          movl      qval, d0              | get Q value
          jsr       hexasc                          | decimal
value to assembly area

          movl      #hdta, a0             | get hit value
address in assembly area
          movl      hitval, d0            | get hit value
          jsr       hexasc                          | decimal
value to assembly area

          movl    #dtamsg, a0              | get start
address of data message
        movl    #dtmse, a1                | get end address
of message + 1
          jsr       prline                          | print
to console
          rts

# Print Line to console.  Call with address of first
character in a0
# register, and the address of the last character + 1 in
the a1 register.
# The routine adds a <CR><LF> at the end of the line.

prline: .word   0x48e7                    | moveml a0-a1, -
```

```
(sp) instruction
        .word   0x00c0                    | mask for a1-a1
        trap    #15                       | invoke monitor
function
        .word   0x0022                    | .outln function
          rts

# Print string to console.  Call with address of first
character in a0
# register, and the address of the last character + 1 in
the a1 register.
# The routine does not add a <CR><LF> at the end of the
string.

prstng: .word   0x48e7                    | moveml a0-a1, -
(sp) instruction
        .word   0x00c0                    | mask for a1-a1
        trap    #15                       | invoke monitor
function
        .word   0x0021                    | .outstr function
          rts


# Read line from console.  Console input is placed in a
buffer until a
# <cr> is typed.  The address of the count byte is returned
in a0.  The
# string consists of the count byte followed by that many
characters.
# The <cr> (or <cr><lf>) are not included in the count of
characters.

rdline:   pea      sdata:1                          | use
data buffer for string
          trap     #15                              | invoke
monitor function
          .word    0x0004                           | .readln
function
          movl     #sdata, a0             | put buffer start
in a0
          rts

# Sonar parameter table.  Each sonar has: (1) the address
```

```
of its data
# register, (2) two ascii characters encoding its number,
and (3) its
# group enable code to write to the sonar command register.

          .even
sptabl:   .long     0xffff83f0          | sonar 0 data
register
          .ascii    " 0"                          |
ascii code
          .word     0x0100                        |
group comand byte
          .long     0xffff83f4          | sonar 1
          .ascii    " 1"
          .word     0x0200
          .long     0xffff83f4          | sonar 2
          .ascii    " 2"
          .word     0x0100
          .long     0xffff83f0          | sonar 3
          .ascii    " 3"
          .word     0x0200
          .long     0xffff83f2          | sonar 4
          .ascii    " 4"
          .word     0x0200
          .long     0xffff83f2          | sonar 5
          .ascii    " 5"
          .word     0x0100
          .long     0xffff83f6          | sonar 6
          .ascii    " 6"
          .word     0x0200
          .long     0xffff83f6          | sonar 7
          .ascii    " 7"
          .word     0x0100
          .long     0xffff83f4          | sonar 8
          .ascii    " 8"
          .word     0x0400
          .long     0xffff83f6          | sonar 9
          .ascii    " 9"
          .word     0x0400
          .long     0xffff83f0          | sonar 10
          .ascii    "10"
          .word     0x0400
          .long     0xffff83f2          | sonar 11
          .ascii    "11"
          .word     0x0400
          .long     0xffff83f0          | min range test
          .ascii    "MN"
          .word     0x0800
          .long     0xffff83f2          | 1K test
          .ascii    "1K"
          .word     0x0800
          .long     0xffff83f4          | 2K test
          .ascii    "2K"
          .word     0x0800
          .long     0xffff83f6          | over test
          .ascii    "OV"
          .word     0x0800

stmesg: .ascii"Press number of sonar to test.  Press Q to
exit"
stmese:
husmsg:   .ascii    "HUSH MODE ON!"
husmse:
gapmsg:   .ascii    "DELAY[mS]=?"
gapmse:
derrms:   .ascii    "BAD DELAY VALUE!"
derrme:
          .even

# Data message assembly area.

dtamsg:   .ascii    " 0 = "
rdta:     .ascii    "    "
          .ascii    ", Q ="
qd  :     .ascii    "    "
          .ascii    ", H ="
hdta:     .ascii    "    "
dtmse:
          .even
gapval:   .long     0                             | current
delay gap value
gapsiz:   .long     0                             |
computed delay gap count
hushf:    .long     0                             | hush
mode flag
saddr:    .long     0                             | sdata
```

```
array pointer
pingct:    .long     0                               | ping
down counter
sdreg:     .long     0                               | data
reg addr for current sonar

nomval:    .long     0                               | nominal
value storage
qval:      .long     0                               | quality
value storage
hitval:    .long     0                               | hit
rate storage

sdata:     .skip     samples * 2          | data value
storage table

endval:    .long     0xaa                            | end of
program mark
```

## APPENDIX B. USER FILE

### 1. Simple Straight Running #1

```
/* Simple Straight Running #1 */
/*Masakuni Michiue June 3, 1994 */
#include "mml.h"
User()
{
CONFIGURATION p1;
int dummy;
def_configuration(0.0, 0.0, 0.0, 0.0, &p1);
speed(10.0);
set_rob(&p1);
line(&p1);
dummy = GetInt();
stop0();
```

# LIST OF REFERENCES

1. 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.

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

3.. Paul Horowitz, Winfield Hill., *The Art Of Electronics-2nd ed.*, New York: Cambridge, 1991

4. Marvin J. Fisher., *Power Electronics,* Boston: PWS-KENT, 1991

5. *Technical Report for LinCMOS Operational Amplifier TLC27x series,* TI Semiconductor Corp., Japan, Tec. Info.No. 74, 1987

6. *Data Sheet of Ultrasonic Transducer T/R40-16,* Japan Ceramic Corp., Japan, 1992

## INITIAL DISTRIBUTION LIST

| | |
|---|---|
| Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145 | 2 |
| Dudley Knox Library, Code 52<br>Naval Postgraduate School<br>Monterey, CA 93943-5101 | 2 |
| Chairman, Code EC<br>Electrical &Computer Engineering Department<br>Naval Postgraduate School<br>Monterey, CA 93943-5121 | 2 |
| Dr. Yutaka Kanayama, Code CS/Ka<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943-5118 | 2 |
| Dr. David Jenn, Code EC/Jn<br>Electrical & Computer Engineering Department<br>Naval Postgraduate School<br>Monterey, CA 93943-5121 | 2 |
| LCDR Masakuni Michiue<br>1st Technical Division, Technical Department<br>Maritime Staff Office, Japanese Defense Agency<br>9-7-45 Akasaka Minato-Ku Tokyo, Japan | 1 |