DESIGN AND IMPLEMENTATION OF
A HIGH SPEED MOBILE ROBOT
FOR TESTING
A NEURAL NETWORK CRASH AVOIDANCE SYSTEM

by

YOICHIRO ENDO

Submitted in partial fulfillment of the requirements

for the Degree of Master of Science

Thesis Advisor: Dr. Roger D. Quinn

Department of Mechanical and Aerospace Engineering

CASE WESTERN RESERVE UNIVERSITY

May, 1998

DESIGN AND IMPLEMENTATION OF A HIGH SPEED MOBILE ROBOT FOR TESTING A NEURAL NETWORK CRASH AVOIDANCE SYSTEM

Abstract

by

YOICHIRO ENDO

Development of a crash avoidance system that can automatically divert an automobile from predicted up-coming collisions is ever more crucial as people rely more on automobiles. The purpose of this thesis is to build a mobile robot to test a neural network crash avoidance system that is based upon the cockroach escape circuit to provide a key to a crash avoidance system for future automobiles.

In this thesis, the robot hardware was integrated with a digital signal processor (DSP) and twelve sonar sensors, and an assembler code was developed to control the robot through the DSP. Since the neural network crash avoidance system is not yet ready for implementation, a simple crash avoidance algorithm, which commands the robot to perform single wall following, was also developed.

The result of the test program proved that both hardware and software of the mobile robot are fully functional and ready for implementation of the neural network crash avoidance system.

DEDICATION

and those who, unfortunately, lost their lives in automobile accidents.

I hope this technology will help to reduce such tragedies in the future.

ACKNOWLEDGEMENT

I would like to greatly thank the following people who supported this thesis.

(In alphabetical order.)

Mr. Nicholas A. Barendt

Dr. Michael S. Branicky

Mr. Mark Dohring

Mr. Chan-Doo Jeong

Mr. Phillip S. Lehmann

Dr. Joseph M. Mansour

Mr. John D. Martens

Dr. Roger D. Quinn

Mr. Yuandao Zhang

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

1.1 Motivation

Because lifestyles of people in modern society heavily rely on motor vehicles, many people lose their lives in traffic accidents. National Highway Traffic Safety Administration (NHTSA) reported that, in 1996, there were 6.8 million traffic crashes in the United States, which killed 41,907 people (NHTSA, 1996).

To prevent loss of life from crashes, installation of seat belts and air bags became standard for automobiles. However, because such technology is not yet commercially available, automobiles are not equipped with any crash avoidance system that can actively divert the vehicles from predicted up-coming collisions.

Numbers of studies have been conducted and methods have been introduced for crash avoidance systems, especially, in the robotics field. Among theses studies, installing a biologically inspired crash avoidance system (Chen, 1996 and Chen et al., 1997), based on a neural network algorithm for the cockroach escape circuit, seems promising for automobiles in the future to perform real-time crash avoidance.

The goal of this thesis is to design and build a mobile robot for implementation of this crash avoidance system, which has been only tested in simulation. It is anticipated that this work will be a key to the implementation of the crash avoidance system on an actual size automobile.

1.2 Related Work

In the field of robotics, it is essential for a robot, whether it is a mobile robot or a stationary industrial robot with a moving manipulator, to be equipped with a crash avoidance or obstacle avoidance system to perform its task in an obstructed environment. Many researchers have studied and developed various approaches for obstacle avoidance including use of a force field, vector field histogram, fuzzy logic, and neural network.

1.2.1 Force Field

One of the classic approaches for obstacle avoidance is to use a force field, or an artificial potential field. In the context of manipulator collision avoidance, for example, the force field approach assumes that the manipulator moves in a field of force; the position of the manipulator to be reached is an attractive pole for the end effector and obstacles are repulsive surfaces for the manipulator parts (Khatib, 1986). Based on the data obtained from a vision sensor, Khatib (1986) achieved real-time obstacle avoidance of a robot manipulator by applying a time-varying artificial potential field. Brooks (1986) and Arkin (1989) applied the force field approach to mobile robots to perform static obstacle avoidance using sonar sensors.

However, the force field approach is only guaranteed to prevent collisions in a static environment (Newman and Hogan, 1987). Furthermore, Koren and Borenstein (1991) found that applying the force field approach on a mobile robot

has four severe limitations; 1.) Trap situations due to local minima; 2.) No passage between closely spaced obstacles; 3.) Oscillations in the presence of obstacles; 4.) Oscillations in narrow passages. Yung and Ye (1996) also pointed out that finding the force coefficients that influence the velocity and direction of the mobile vehicle is difficult in a cluttered environment which is too complex to be embedded in a mathematical model.

1.2.2 Vector Field Histogram

A useful approach for a mobile robot to perform obstacle avoidance is the Vector Field Histogram (VFH) method that utilizes a two-dimensional Cartesian Histogram Grid. Based on the continuously updated data from onboard sonar sensors, Borenstein and Koren (1990) employed the VFH method to designate the certainty value in each cell of the Histogram Grid as the confidence of the algorithm in the existence of an obstacle at that location. After converting the two-dimensional Histogram Grid into a one-dimensional Polar Histogram, which represents the polar obstacle density around the robot, the robot performs obstacle avoidance by moving towards the sector that contains low obstacle density and is close to the target. This method allows a robot to navigate through a narrow passage without oscillations. Borenstein and Koren (1991) integrated the VFH method into a mobile robot with 24 sonar sensors onboard, and could perform real-time obstacle avoidance with an average speed of 0.54 m/s (1.77 ft/s).

Gourley and Trivedi (1994) also achieved real-time obstacle avoidance by

implementing a simplified version of the VFH method on a mobile robot, ELVIS, with 16 sonar sensors onboard. Instead of using the occupancy grids, the heading direction of ELVIS was determined by a sum of a desired direction and a resultant of the 16 weighted vectors that were calculated from the inverted reading vectors of the sonar sensors times their associated weights. The average speeds of the robot were 0.32 m/s (1.05 ft/s) in a hallway and 0.17 m/s (0.56 ft/s) in a cluttered lab.

1.2.3 Fuzzy Logic

Use of fuzzy logic, is another alternative approach for a mobile robot to perform crash avoidance. The nature of fuzzy logic is highly mathematical, and it can provide a robust and consistent foundation for information processing, including pattern-formatted information processing (Pao, 1989).

Yung and Ye (1996) developed a self-learning fuzzy navigation method, which utilizes fuzzy logic and reinforcement learning; the fuzzy rules of the on-line obstacle avoidance of a mobile vehicle are learned through reinforcement learning. The advantages of this method are high learning speed, high number of learned rules, high adaptability, and reliable convergence of the learning network. This method was verified, in a simulation program, as an effective learning method for the mobile vehicle to perform obstacle avoidance.

The fuzzy control system that Ming et al. (1995) developed also allows a mobile robot to avoid unexpected obstacles in a partially unknown environment.

The fuzzy control system is a rule-based system that utilizes fuzzy linguistic variables to model rules given by a human. Membership functions translate the selected fuzzy linguistic variables into the precise numeric values needed by a computer. To optimize the system, genetic algorithms, a search technique analogous to evolution, were used to select the best membership functions for the fuzzy control system. In a simulation program, the fuzzy control system caused the mobile robot, equipped with sonar sensors, to perform wall-following obstacle avoidance.

1.2.4 Neural Network

The biologically inspired crash avoidance system, described in section 1.3, is a neural network based upon the cockroach escape circuit. Employing a neural network algorithm seems to be a promising approach for obstacle avoidance of a mobile robot. The advantage of the neural network is its ability to learn, and it can be implemented rather easily on a microcomputer (Touretzky and Pomerleau, 1989).

An interesting project that verified the power of the neural network is Autonomous Land Vehicle In a Neural Network (ALVINN), developed by Pomerleau (1993). ALVINN was implemented on an automobile utilizing the backpropagation method to perform autonomous road following. The neural network consists of three layers: an input layer of a two-dimensional image from a video camera or scanning laser rangefinder, a hidden layer, and an output layer

of a vector of units representing different steering responses. ALVINN trains the network by observing a person driving the vehicle. According to the latest record quoted in the web site[*] of this project, ALVINN has successfully driven autonomously at speeds of up to 70 mph for over 90 miles of a multilane highway after about three minutes of observing a person driving. ALVINN also performed static obstacle avoidance by employing the laser range images. However, due to the slow sampling rate, two images per second, of the laser rangefinder, the obstacle avoidance could not have been achieved at high speed.

Schiller and Tench (1989) also demonstrated the strength of neural networks by applying the backpropagation method to the guidance of an autonomous underwater vehicle (AUV). This network also consists of three layers. The inputs of the guidance system are the readings of nine onboard sonar sensors and the difference between the current AUV course and the straight-line course required to reach the goal point. A course correction, how far left or right of the current course the AUV should be directed, is the output. Even though it was only implemented in a simulation program, after a few hundred steps of training, the AUV learned the way to avoid obstacles and navigate towards the goal.

1.3 Chen's Crash Avoidance System

Chun-Ta Chen in collaboration with Roger D. Quinn and Roy E. Ritzmann developed a crash avoidance system for automobiles as his Ph.D. dissertation

[*] http://www.cs.cmu.edu/afs/cs.cmu.edu/project/alv/member/www/projects/ALVINN.html

(1996) in the Bio-Robotics Laboratory, Case Western Reserve University (Chen at al., 1997). Chen's crash avoidance system utilizes a neural network algorithm based upon a distributed network of artificial neurons that mimic the neural organization of the escape circuit in the American cockroach, *Periplaneta americana*. A neural network algorithm for the cockroach escape circuit was originally developed by Beer and Chiel (1993) based upon results of a research conducted on the cockroach in the Ritzmann Laboratory, Case Western Reserve University.

The cockroach detects wind caused by an approaching predator with sensitive hairs located on its cerci, two antenna-like appendages on the rear of its abdomen. Based on the wind information, the thoracic interneurons send signals to the leg motor neurons to move the cockroach away from the predator (Ritzmann, 1993). The cockroach can perform this escape response very rapidly, in approximately 60 ms. Ritzmann believed that the cockroach escape circuit satisfies all of the requirements for a successful crash avoidance system for automobiles.

The neural network algorithm of Chen's crash avoidance system utilizes the backpropagation method. Layers of artificial neurons learn the pattern of input-output training sets through many iterations, and store the pattern into connection weights. Based on the weights of the recognized pattern, the network can rapidly compute an output from a newly fed input. This method allows the crash avoidance system to learn the pattern of a crash avoidance strategy off-line, and provide on-line real-time collision avoidance control to the vehicle.

The control of Chen's crash avoidance system is semiautonomous. A driver can operate the vehicle normally unless a collision is foreseen. When the up-coming collision is predicted, the crash avoidance system takes over the driver's control of the vehicle, and attempts to avoid the collision by steering, accelerating, or braking. The prediction of the collision, crash alarm set, is based on the present state of the obstacle with respect to the vehicle. Onboard sonar sensors detect the location of the obstacle as well as its heading speed and direction. The current and previous readings of the sonar sensors are sent to the crash avoidance system, so that, along with the current speed and steering angle of the vehicle, it can calculate the present state of the obstacle. The system was shown to be very successful in simulation.

1.4 ROACH

In order to implement Chen's crash avoidance system, a mobile robot, ROACH* (Fig. 1.1), was built at the Center for Automation and Intelligent Systems Research, Case Western Reserve University. The hardware and software features of the robot are reported in this thesis. The hardware aspect is described in Chapter II. Chapter III deals with the software aspect. While modification of the algorithm for Chen's crash avoidance system is in progress to be retrofitted into the robot, ROACH was tested with a simple obstacle avoidance algorithm

* The **R**obot with an **O**bstacle **A**voidance System and **C**ontroller for **H**igh Speed Mobility

that could check the capability of the robot. Implementation of the test program is described in Chapter IV. The conclusions, discussion, and future work of this project are in Chapter V.

CHAPTER II

MOBILE ROBOT HARDWARE

2.1 Hardware Criteria

Before ROACH was built, in order for the robot to achieve its mission properly, the following three points were considered as criteria to construct the hardware of the robot:

1.) mobility

2.) agility

3.) durability

2.1.1 Mobility

Since the robot is meant to simulate the characteristic of an automobile, it is mobilized with four wheels driven by an electric motor. In order to provide the robot full mobility, it is also important that no cables are attached to the robot from any stationary device: a power-plug in a wall or a serial port of a personal computer, for example.

2.1.2 Agility

The robot had to be able to run at high speeds to simulate scaled automobile highway speeds. In order to produce such high speed, the robot is fabricated with many of the light components from a radio controlled model car.

2.1.3 Durability

Numbers of collisions are expected to occur during debugging of the system because the purpose of the robot is to test the crash avoidance system, which has never been tested besides in simulation. Thus, the robot has to be durable to the shocks caused by collisions.

2.2 Hardware Overview

ROACH is a four-wheel mobile robot that is fabricated with many of the components from a 1:10 scale radio controlled (RC) car, Tamiya Blackfoot Ford F-150 Ranger. The frame of the RC car was reconstructed to mount twelve sonar sensors, a digital signal processing (DSP) board, and electric circuit boards on the robot. A front bumper was also added to the robot to reduce the shock in case of collisions during operation.

The size of ROACH is 18.6-inch long, 12.5-inch wide, and 10.0-inch tall (Appendix A). The robot weighs 10 pounds, and has a 10 ft/s maximum speed and 13° maximum steering angle to both the left and right.

ROACH was designed to be for in-lab use only, and it does not resist water or dirt. Since it is going to be tested only on the flat surface of the lab, it has a rigid suspension.

2.3 Components Description

As they are also labeled in Fig. 2.1 and listed in Appendix B, ROACH consists of the following components[*]:

1.) receiver (1)

2.) servo (1)

3.) speed controller (1)

4.) motor (1)

5.) sonar sensors (12)

6.) DSP board (1)

7.) electric circuit boards (2)

8.) batteries (2)

9.) wheels (4)

10.) frame (1)

11.) bumper (1)

2.3.1 Receiver

The receiver, Futaba FP-R112JE, is equipped with a built-in pulse modulator. It receives radio signals, carrying data for desired speed (v) and steering angle (ϕ), from a transmitter. It converts the radio signals into two channels (CH1 and CH2) of 50 Hz 5-volt pulses. As shown in Fig. 2.2, in the case of an RC car, the 50 Hz pulses in CH1 are sent directly to the servo, and the pulses in CH2 are sent to the

[*] Numbers in brackets indicate quantity.

speed controller. In case of ROACH, on the other hand, the 50 Hz pulses from the receiver are intercepted by the DSP as shown in Fig. 2.3. The DSP also receives signals from the sonar sensors. The DSP generates its own 50 Hz pulses based on the crash avoidance algorithm, and sends them to the servo and speed controller.

2.3.2 Servo

When the servo, Futaba FP-S148, receives the 50 Hz pulses, it revolves a servo horn, the rotation part of the servo, connected to the front wheels of the robot through rods (Fig. 2.4). When the servo horn rotates, the front wheels also turn. The rotation angle of the servo horn is determined by the pulse width of each pulse (Fig. 2.5). From observation, when the pulse width sent to the servo is 1.42 ms, the steering angle of the front wheels reaches its maximum to the right, 13° clockwise. The front wheels turns the maximum steering angle to the left, 13° counterclockwise, when the pulse width is 0.86 ms.

2.3.3 Speed Controller

The speed controller, Dynamite Power Pulse Speed Control (1996), sends current to the motor when it receives the 50 Hz pulses. The amount of current is proportional to the input pulse width. Thus, the speed of the motor is also proportional to the pulse width. When the pulse width is 1.14 ms, the speed controller sends the maximum current, 180 A, and the motor spins forward with its maximum speed. The motor stops spinning when the pulse width is 1.42 ms.

When the pulse width is more than 1.42 ms, the motor spins backwards.

2.3.4 Motor

The 7.2-volt DC brush motor drives the rear wheels of the robot with respect to the amount of the current received from the speed controller. The motor is manufactured by Johnson; however, the data of Johnson's motor was not available. The characteristics of the motor were estimated from the same type of motor, Mabuchi Motor RS 540SH. As the graph in Fig. 2.6 shows, this type of motor can produce 1,900 g·cm (0.186 N·m) maximum torque when 50 A current is applied, and can spin as fast as 16,000 rpm with no load.

2.3.5 Sonar Sensors

An array of sonar sensors is mounted on top of the robot platform to measure the distance between the robot and surrounding obstacles. It sits 9.2 inches high from the ground. Each sonar sensor is fabricated from a transducer, Polaroid 600 Series Instrument Grade Electrostatic Transducer, and an ultrasonic circuit board, Polaroid 6500 Series Sonar Ranging Module (1991).

When the initiate input (INIT) pin on the circuit board is raised from LO (0-volt) to HI (5-volt), the transducer fires 16 cycles of sonar pulse-signals at a frequency of 49.4 kHz. When the transducer detects the returning signals bounced back from an obstacle, the circuit board sets the echo output (ECHO) pin to HI (Fig. 2.7).

The output from the sonar sensor does not include a direct measurement of time. Thus, the internal timer of the DSP is used to measure the time-of-flight or duration of the sonar pulse-signals. The time-of-flight of the sonar pulse-signals is taken from the time difference (/t) between when the INIT pin is raised to HI and when the ECHO pin is raised to HI (Fig. 2.8).

The distance (r) between the robot and the obstacle is one half of the path length that the sonar pulse-signals travel with the speed of sound (c):

$$r\,(\text{ft}) = \frac{c\,(\text{ft/ms}) \times \delta t\,(\text{ms})}{2} \qquad \text{Eqn. 2.1}$$

In order to avoid detecting the pulse-signals bounced back from its own components instead of the obstacle, the sonar sensor can disable the detection of the pulse-signals during a blanking time. The blanking time is pre-configured for 2.38 ms by the manufacturer, and can be reconfigured by sending input signals to the blanking (BLNK) pin and blanking inhibit (BINH) pin on the ultrasonic circuit board. When the voltage on the BLNK pin is set to HI, the sonar sensor does not detect the incoming pulse-signals from the transducer. The BINH pin has to be raised to HI prior to the blanking time in order to activate the BLNK pin. If the obstacle needs to be detected earlier than the pre-configured blanking time, 2.38 ms, as it is shown in Fig. 2.9, the BLNK pin and BINH pin should be kept LO until a desired reading time starts. When the desired reading time begins, the BINH pin should be set to HI while keeping the BLNK pin LO. The blanking time for ROACH is configured with this setup, and it can detect obstacles as close

as 6 inches.

The maximum distance of the sonar sensor being able to detect an obstacle is approximately 35 feet. However, due to the time reserved for the sonar sensor routine in the control algorithm, the maximum distance that ROACH is able to detect an obstacle is 7 feet. The time constraint for the control algorithm is explained in section 3.2.

As it is shown in Fig. 2.10, each transducer transmits its sonar pulse-signals in a 30° angle cone. Thus, in order for the robot to scan the 360° horizontal direction, twelve sonar sensors are mounted to form a circular array (Fig. 2.11).

The sonar sensors are numbered from 0 to 11 (Fig. 2.12). The robot can simultaneously fire two sonar sensors that are 180° apart, and six sonar sets are alternatively fired to complete the entire 360° scan. The first sonar set, Sonar-Set 1, is the pair of Sonar 0 and 6; Sonar-Set 2, 3, 4, 5, and 6 are pairs of Sonar 3 - 9, 1 - 7, 4 - 10, 2 - 8, and 5 - 11, respectively. When more than two sonar sensors detect obstacles at the same time, the robot concentrates on the closest one.

2.3.6 DSP Board

The DSP board is analogous to the nerve center of the robot. The function of the DSP board is to control the actuators, activate the sonar sensors, and compute proper values for the speed and steering angle of the robot to avoid obstacles.

The DSP board, Analog Devices ADSP-2181 (1995) EZ-KIT Lite (1995), shown in Fig. 2.13, has the following features that are utilized by ROACH:

1.) digital signal processor (DSP)

2.) random access memory (RAM)

3.) expansion connectors

2.3.6.1 DSP

The 16-bit fixed-point DSP, which is integrated into the DSP board, executes instructions of an executable (EXE) machine code in the RAM with an instruction cycle-time of 30 ns (33 MHz). The EXE file is created from a code written in assembly language using the Assembler and Linker (Analog Devices, 1994). A code written in C language can also be converted into assembly language using the C Compiler (Analog Devices, 1994). Thus, the crash avoidance algorithm written in C language can be converted into assembly language, so that it can be linked to the EXE file. ADSP-2181 was designed for 16-bit fixed-point signal processing, so that it slows down the performance when it is processing the 32-bit floating-point data. Thus, floating-point data in the program should be converted into fixed-point data if possible.

The EXE file can be downloaded from a personal computer (PC) through the serial port by running the EZ-KIT Lite Host Program (Analog Devices). Since the serial cable can be detached from the serial port connector on the DSP board after downloading, it allows the robot to operate without any cables attached from the PC.

Because the assembler code is written in a low-level language, not only can it

perform simple calculations, but also it can modulate the digital signals of the expansion connectors on the DSP board. This feature allows the DSP to control other components of the robot, such as the servo and speed controller. The control algorithm developed for ROACH is described in section 3.2.

2.3.6.2 RAM

The EXE file stored in the RAM is divided and allocated to 24-bit program memory (PM) and 16-bit data memory (DM) segments. The PM and DM can be downloaded either together or separately. The RAM can store 16K words (48 KB) PM and 16K words (32 KB) DM. The maximum size of downloadable PM is, however, about 14K words because about 2K words of PM are reserved for the host program to link the DSP to the PC.

When the power for the DSP board is turned off, the memory in the RAM clears its contents, and the program needs to be loaded into the RAM each power-up reset. The program can be downloaded from the PC through the serial port connector each power-up reset, or from an erasable-programmable read-only memory (EPROM), which can permanently store the program inside the DSP board.

The Prom Splitter (Analog Devices) creates a programmable read-only (PROM) file from the EXE file. With an EPROM programmer, the PROM file can be programmed into the EPROM. The DSP has a socket into which the EPROM can be inserted, and the contents of the EPROM can be loaded into the

RAM upon the power-up reset or when the reset button on the DSP board is pushed. The contents of the EPROM can be erased only by exposure to ultraviolet light.

Nevertheless, the EPROM is not utilized to store the program in this project because numbers of experimental programs are to be tested in the robot, and quick changeover of the programs is preferred.

2.3.6.3 Expansion Connectors

Two sets of the 50-pin expansion connectors (P2 and P3) allow the DSP board to interface with external components. Each pin is assigned to a DSP signal, and the name of each signal is listed in Tables 2.1 and 2.2. Out of the 50 pins of P3, the DSP board utilizes three flag output pins and eight programmable flag pins to communicate with other components of the robot.

The DSP board can send digital output signals from the flag output pins (FL0, FL1, and FL2). FL0 and FL2 send the signals to the servo and speed controller, respectively. FL1 is internally hooked up to the red LED on the DSP board, and is mainly used for debugging the assembler codes. By hooking up FL1 to an alarm-set, it can be also used for the robot to give warning signals when an up-coming collision is predicted by the crash avoidance system.

The programmable flag pins (from PF0 to PF7) can be programmed to be either input or output pins. PF0 sends the BINH signals to the BINH pin on the ultrasonic circuit board. PF1 and PF2 receive the CH1 and CH2 pulses from the

receiver, respectively. After two sonar sensors fire the sonar pulses-signals simultaneously, PF3 reads the ECHO signal (ECHO1) from one of the sonar sensor, and PF4 receives ECHO2 from the other sonar sensor. The rest of the programmable flag pins (PF5, PF6, and PF7) are hooked up to the sonar interface circuit board that selects the set of sonar sensors and sends the INIT signals. The sonar interface circuit board is explained in the next section (2.3.7).

2.3.7 Electric Circuit Boards

Two electric circuit boards (Board-A and Board-B) were built to equip the robot with the following features:

1.) interface for the DSP board to the servo

2.) interface for the DSP board to the speed controller

3.) interface for the DSP board to CH1

4.) interface for the DSP board to CH2

5.) interface for the DSP board to the alarm set (for future use)

6.) interface for the DSP board to the ultrasonic circuit board

7.) voltage regulator

The sixth feature, the sonar interface circuit, is divided into two parts and integrated into both Board-A and Board-B. The rest of the features above are integrated into only Board-A. The schematic shown in Fig. 2.14 illustrates how the DSP board is hooked up to other components.

The sonar interface circuit was modified from the one on Martens' Andros

Mark VI (Martens, 1993). The primary function of the sonar interface circuit is to reduce the number of the input and output (I/O) signals modulated by the DSP board, which has limited I/O pins. Since twelve sonar sensors are mounted on ROACH, and each sonar sensor has three input (INIT, BLNK, and BINH) pins and one output (ECHO) pin, a total of 48 I/O signals are required to operate all twelve sonar sensors. As it is shown in Fig. 2.15, the sonar interface circuit consists of a demultiplexer (74LS138), 24 tristate buffers (74HCT125), and 12 pull-up registers (47 kΩ). Three DSP programmable flag pins (PF5, PF6, and PF7) are connected to three input pins of the demultiplexer. As it is shown in Table 2.3, based on the combination of the enabled input pins, the demultiplexer selects one of six output pins[*], which corresponds to one of six sonar sets, and enables four tristate buffers. Each ultrasonic circuit board receives the INIT signal from a tristate buffer and returns the ECHO signal to another tristate buffer (Fig. 2.16). The demultiplexer is integrated into Board-A, and the tristate buffers are integrated into Board-B.

The demultiplexer, tristate buffers, and ultrasonic circuit boards need a 5-volt power supply. Since the installed 6-cell batteries generate 7.2-volt powers, two voltage regulators (MC7805C) are integrated into Board-A to reduce the voltage (Fig. 2.17). One of the 5-volt outputs from the regulators supplies power to the demultiplexer and tristate buffers. The other 5-volt output supplies power to the ultrasonic circuit boards.

[*] The demultiplexer itself is capable of selecting eight output pins.

The two electric circuit boards were made from single-sided, positive-type etching boards. The layouts used for the etching are shown in Figs. 2.18 and 2.19.

2.3.8 Batteries

Two 6-cell (7.2-volt) rechargeable batteries, Dynamite DYNA-SPORT 1500, supply power to the robot. One of the batteries, Battery-A, is hooked up to the speed controller, and supplies power to the speed controller, receiver, servo, and motor. The other battery, Battery-B, is hooked up to Board-A, and supplies power to the circuits in Board-A, Board-B, the DSP board, and the ultrasonic circuit boards.

Since the DSP is sensitive to electrical noises from other components and the receiver produces relatively large noises, two separate batteries are used, instead of just one, to stabilize the circuits of the DSP board.

Each battery lasts approximately 5 minutes when it is used full time, and it requires a charging time of 20 minutes. However, external power sources can replace Battery-B when the robot is tested in a stationary position. Board-A can take 5-volt DC power externally and supply the power to Board-A, Board-B, and the ultrasonic circuit boards. The power for the DSP board can be also supplied by an adapter that converts 120-volt AC to 9-volt DC power.

2.3.9 Wheels

ROACH is mobilized with four wheels to simulate the characteristics of an automobile. The front two wheels are connected to the servo for steering. They can produce the maximum steering angle of 13° both to the left and right. As shown in Fig. 2.20, the rear two wheels are connected to the motor through four gears. Gear-1, fixed to the motor shaft, has 10 teeth meshing with 52 teeth of Gear-2. Gear-2 is fixed to Gear-3, and 17 teeth of Gear-3 meshes with 48 teeth of Gear-4. Gear-4 drives the shaft of the rear wheels. This configuration produces one revolution of the rear wheels from approximately 15 revolutions of the motor.

2.3.10 Frame

In order for ROACH to test high speed crash avoidance, the robot should be as light as possible. Thus, the frame of the robot was made of aluminum, a lightweight metal. Another advantage of aluminum is its softness for ease of machining.

As it is shown in Fig. 2.21, the frame of ROACH holds three platforms. The bottom platform is for mounting the servo, speed controller, batteries, motor, gearbox, four wheels, and bumper. The middle platform is for the DSP board and Board-A. The twelve sonar sensors and Board-B are mounted on the top platform.

2.3.11 Bumper

Since the mission of ROACH is to test the crash avoidance algorithm that has

never been tested, besides in a simulation program, errors may lead the robot to collide with obstacles instead of avoiding them. Shocks of those collisions can damage the structure or the electric components of the robot. The bumper, shown in Fig. 2.22, was designed to reduce such damage. The bumper consists of a mount, a pair of springs, a bumper-frame, and rubber foam.

The mount is fixed to the frame of the robot, and the pair of the springs sits between the mount and the bumper-frame. When the bumper-frame contacts the obstacle, the pre-collision kinetic energy is transformed into the post-collision potential energy of the springs while being compressed. From the conservation of energy, Eqn. 2.2 formulates the relationship among the distance (x) that the springs are compressed, the total stiffness (k) of the springs, the mass (m) of the robot, and the relative velocity (v) between the robot and the obstacle right before the collision:

$$x\,(\text{in}) = \sqrt{\frac{12\,(\text{in/ft}) \times m\,(\text{slug}) \times v^2\,(\text{ft}^2/\text{s}^2)}{k\,(\text{lb/in})}} \qquad \text{Eqn. 2.2}$$

The robot weighs 10 pounds (0.311 slugs). Each spring has a stiffness of 26.7 lb/in (53.4 lb/in for the total stiffness) and length of 3.5 inches, which can be compressed to a half of the size (1.75 inches). Thus, the springs are compressed all the way when the relative velocity before the collision is 6.6 ft/s. The actual relative velocity, however, may be expected to be slightly larger because the rubber foam glued to the bumper-frame also absorbs energy.

CHAPTER III

MOBILE ROBOT SOFTWARE

3.1 Software Overview

The DSP board, the nerve center of the robot, controls the actuators, activates the sonar sensors, and computes proper speed and steering angle of the robot. These tasks are all managed by an execution program downloaded into the RAM of the DSP board. The execution program is divided into two parts: a control algorithm, which manages the low-level tasks of the robot, and a crash avoidance algorithm, which computes the proper speed and steering angle of the robot from the data given by the control algorithm.

Currently, modification of Chen's crash avoidance system is in progress to be retrofitted into ROACH. Thus, a simple crash avoidance algorithm, which is a substitute for Chen's crash avoidance, was developed to test the hardware of the robot.

3.2 Control Algorithm

The control algorithm for ROACH, coded with approximately 500 lines, is written in assembly language (Appendix C). The low-level tasks that the control algorithm manages include:

1.) Read the CH1 and CH2 pulses from the receiver.

2.) Send the INIT signals to the sonar sensors.

3.) Read the ECHO signals from the sonar sensors.

4.) Provide the input values to the crash avoidance algorithm.

5.) Receive the output values from the crash avoidance algorithm.

6.) Send the pulses to the servo and speed controller.

As explained in section 2.3.1, the speed controller and servo are specifically configured for 50 Hz inputs. Hence, it is essential for them to receive pulses from the DSP board every 20 ms. In order to execute the tasks above with this time constraint, a timing schedule for the control algorithm was developed. The flow chart in Fig. 3.1 illustrates the schedule, and is used to describe the structure of the control algorithm in terms of the routines below.

3.2.1 Routines

Each white box in Fig. 3.1 indicates a routine of the control algorithm. As shown in Appendix C, the code of the control algorithm is divided into several segments in terms of tasks they deal with. Each segment is defined as a routine in this thesis. The routines are executed sequentially after being loaded into RAM. The routines between Timer-Reset and Stand-By are repeated every 20 ms to form a loop. In Fig. 3.1, a big gray box, Selection Block, is a group of routines from which a different set of routines is selected and executed for each 20 ms loop.

3.2.1.1 Timer-Setup

The first routine, Timer-Setup, handles an initial setup for the DSP timing configuration. It sets up the timing variables and enables the timer. Details on the timer configuration are explained in section 3.2.2. This routine, labeled as “start”, is pointed by the reset interrupt vector address (Table 3.1). Thus, it is executed as soon as the program is loaded into RAM. However, this routine never ends because of the infinite loop, Code 3.1, attached at the end of the routine:

```
wait:   NOP;
        jump wait;
```

Code 3.1

The purpose of this infinite loop is to wait for the timer interrupts before the program finishes the execution.

3.2.1.2 Timer-Reset

The second routine, Timer-Reset, resets the clocks that measure the pulse widths of CH1 (input), CH2 (input), the servo (output), the speed controller (output), and the time-of-flight of the sonar pulse-signals (input). This routine begins when the PF1 pin, connected to CH1, is set to HI. If there is no pulse coming from CH1, this routine can not be executed, and the rest of the program will remain idle. Thus, in case of emergency, when the robot needs to be stopped, an operator can simply turn off the power of the transmitter, so that the pulse will not be sent to CH1.

3.2.1.3 Pulse-In

The third routine, Pulse-In, starts just after Timer-Reset is executed. At first, it measures the pulse width of the CH1 pulse by polling the PF1 pin. As soon as the PF1 pin perceives the falling edge of the CH1 pulse, the PF2 pin starts measuring the pulse width of the CH2 pulse, which always comes right after the CH1 pulse. Pulse-In has 3.5 ms execution time to complete its task.

3.2.1.4 Selection Block

After the widths of the CH1 and CH2 pulses are recorded in Pulse-In, the next routine is selected from Selection Block. Selection Block either activates the sonar sensors and records the readings, or runs the crash avoidance algorithm.

Sonar-INIT activates the sonar sensor by sending the INIT input to a sonar set. At each 20 ms loop, one set is chosen from the six sonar sets. Since the six sonar sets are chosen sequentially with the loops, it takes 120 ms for them to complete the 360° scan. Sonar-INIT is followed by Sonar-ECHO that reads the ECHO output and records the time-of-flight of the sonar pulse-signals. An execution time of 12.6 ms is reserved for Selection Block, so that the sonar sensor can detect obstacles as far away as 7ft.

After the 360° scan by the sonar sensors is completed, at the next 20 ms loop, Selection Block runs the crash avoidance algorithm to compute the speed and steering angle of the robot. Thus, the minimum cycle time that the control algorithm can update the output values from the crash avoidance algorithm is 140

ms.

3.2.1.5 Pulse-Out

After the routines in Selection Block are completed, Pulse-Out sends pulses to the servo and speed controller. It raises the voltage on the FL0 and FL2 pins, connected to the servo and speed controller, respectively, to HI until proper pulse widths are sent. The execution time for this routine varies with the pulse widths.

3.2.1.6 Stand-By

After Pulse-Out completes delivery of the output pulses to the servo and controller, the last routine, Stand-By, attempts to find the rising edge of next CH1 input pulse by constantly polling the PF1 pin. When the pin is set to HI, the execution program goes back to Timer-Reset, the beginning of the 20 ms loop.

3.2.2 DSP Timer Configuration

A programmable internal timer is integrated into the DSP board, and it can generate periodic interrupts to the execution program. When the execution program receives the interrupt from the timer, it shifts its execution address to the timer interrupt vector address, the eleventh row of the interrupt vector table (Table 3.1), and starts executing the instruction of the address.

The interval of the timer interrupts can be configured by a combination of an 8-bit prescaler register (TSTEP) and a 16-bit count register (TLENGTH). TSTEP

scales the length of a time unit that TLENGTH counts. In the control program, the value of TSTEP is 40, which corresponds to 1,200 nanoseconds[*], and the value of TLENGTH is 10. Each time when TLENGTH counts TSTEP ten times, taking 0.012 ms, the execution address of the execution program jumps to the timer interrupt vector address. As it is shown in Table 3.1, the instruction of the control algorithm at the timer interrupt vector address is "jump update_signal". The label "update_signal" is located at the first line of Pulse-In, and the first portion of Pulse-In determines current status of the program in terms of the time schedule for the 20 ms loop. For example, if current time is determined to be between 0 and 3.5 ms, the program executes the rest of the instructions in Pulse-In; otherwise, it skips them, and executes the instructions in an appropriate routine.

Acquiring or producing resolution of the pulses is defined by the interval of the timer interrupts since it determines how often the pulses are polled or updated. The interval of the timer interrupts for ROACH, 0.012 ms, allows the robot to have the resolution of input or output as fine as approximately 0.6° for the steering angle or approximately 0.9 ft/s for the speed.

3.3 Simple Crash Avoidance Algorithm

The purpose of the simple crash avoidance algorithm is to substitute for Chen's crash avoidance system, which is still in progress to be retrofitted into ROACH. Thus, this algorithm is not meant to perform perfect crash avoidance.

[*] 40×30 ns (a cycle-time of the DSP) = 1,200 ns

The current problem with implementation of Chen's crash avoidance system is discussed in section 5.2.2.

The simple crash avoidance algorithm is for ROACH to perform single wall following, or "a drunken sailor walk" (Arkin, 1989). The robot tries to follow a wall by maintaining a distance from it (Fig. 3.2). Instead of all twelve sonar sensors, only three sonar sensors, Sonar 1 to 3, are activated. When one of the sensors detects the wall within two feet on its right, the robot turns left, 13°, to stay away from the wall. When there is no wall within two feet, on the other hand, the robot turns right, 13°, to find the wall. In order to minimize the risk of a collision, the speed of the robot is directly controlled by an operator.

There are two versions of the simple crash avoidance algorithm: one written in C language (Appendix D), and the other one written in assembly language (Appendix E). They were both tested on ROACH to check the compatibility of the control algorithm (Chapter IV). The one written in assembly language was originally produced with the C Compiler by converting the one written in C language. The code that the C Compiler converted was produced as an external program (Appendix F). Thus, in order for the code to be pasted into the body of the control algorithm, a portion of the code that dealt with the interface between a host and external program, such as a function prologue and epilogue, was deleted. On the other hand, when the crash avoidance is called as an external program written in C language, an algorithm (Appendix G) that interfaces the assembly and C languages has to be added to the control algorithm.

CHAPTER IV

IMPLEMENTATION

4.1 Test Program

A test program that consists of the control algorithm and the simple crash avoidance algorithm was implemented into ROACH to check the performance of the components. Both versions, in C and assembly languages, of the simple crash avoidance were tried with the test program to check the compatibility of the control algorithm.

4.2 Result

All components of ROACH including the receiver, servo, speed controller, motor, DSP board, sonar sensors, and electric circuit boards worked nominally. No malfunction was found in the hardware.

The simple crash avoidance in both C and assembly languages produced the same behavior of the robot. As shown in Fig. 4.1, ROACH with the simple crash avoidance was able to follow a straight wall in a hallway even though the robot had a tendency to oscillate its heading direction. ROACH was also able to make a 90° right turn at the edge of the wall (Fig. 4.2). However, ROACH had difficulty in making a 90° left turn at a corner of two walls unless the angle was reduced (Fig. 4.3).

ROACH could perform the simple crash avoidance above only at relatively

low speed. The average speed that ROACH could comfortably achieve the straight wall following was approximately 2.9 ft/s. This is faster than Borenstein and Koren's mobile robot (1991), which performed obstacle avoidance with an average speed of 1.77 ft/s by employing the VFH method. Without any crash avoidance, however, ROACH is capable of running up to 10 ft/s.

CHAPTER V

CONCLUSIONS, DISCUSSION, AND FUTURE WORK

5.1 Conclusions

ROACH, a four-wheel mobile robot designed to test a biologically inspired crash avoidance system, was successfully built from both hardware and software aspects. Integrated components of ROACH include a receiver, a servo, a speed controller, a motor, twelve sonar sensors, a DSP board, two electric circuit boards, two batteries, four wheels, a frame, and a bumper. ROACH is capable of running at speeds up to 10 ft/s. During the 360° scanning by the sonar sensors, ROACH can detect an obstacle as far away as 7 ft, and as close as 6 inches. ROACH updates the information of the obstacle every 140 ms. A control algorithm was developed for the DSP to control the actuators and activate the sonar sensors. In order to substitute for Chen's crash avoidance system, which is not yet ready for implementation, a simple crash avoidance algorithm that commands the robot, using the data from the control algorithm, to perform single wall following was also developed. The result of a test program, which consists of the control algorithm and the simple crash avoidance algorithm, proved that the robot is fully functional and ready for implementation of Chen's crash avoidance system.

5.2 Discussion

5.2.1 Design Criteria and ROACH Hardware

The three design criteria, mobility, agility, and durability, which were considered for construction of the robot hardware (section 2.1), were all satisfied by ROACH.

5.2.1.1. Mobility

Since ROACH is fabricated with the parts from a 1:10 scale four-wheel RC car, an operator can maneuver the robot wirelessly. Furthermore, all the low-level tasks of the control algorithm as well as all the computations necessary for the crash avoidance algorithm can be achieved by an executable program downloaded into the RAM of the DSP board. This feature allows ROACH to perform its tasks without any cables attached from external computers.

5.2.1.2. Agility

ROACH is capable of running at speeds up to 10 ft/s. Thus, ROACH is capable of simulating an automobile driving on a highway with a speed of 65 mph (95.3 ft/s), which is 9.5 ft/s[*] for the 1:10 scale robot.

[*] Assuming straightforward scaling by length.

5.2.1.3. Durability

A front bumper was constructed and installed on ROACH to protect the components from unexpected collisions. For absorption of the shock caused by a collision, the springs of the bumper are compressed all the way when the relative velocity of the robot with respect to the obstacle before the collision is 6.6 ft/s.

5.2.2 Current Software Problem

The current problem causing the delay of implementing is, however, a software problem. Even though the control algorithm was proved to be compatible with the crash avoidance algorithm written in C language, it was found that the original version of Chen's crash avoidance algorithm, written in C language, was too large and too complicated for the DSP. For example, the original version of Chen's crash avoidance system required 21k words program memory while the RAM can store the total program memory of 16K words. Several modifications of Chen's crash avoidance algorithm were attempted by Chan-Doo Jeong, a Ph.D. candidate at Case Western Reserve University, and myself. The modifications include:

1.) *Changing most of the floating-point numbers to fixed-point numbers*: Since the DSP was built for 16-bit signal processing, floating-point operations require much larger program memory than fixed-point operations.

2.) *Reducing the number of subroutines*: Calling many subroutines was

presumed to be the cause of a stack memory overflow, which frequently halted the program.

3.) *Rewriting the entire program in assembly language*: Since assembly language is a low-level language, not only can it provide better understanding of the machine-level operations, but also, when the program is halted, the location of the problem can be traced much easier than in C language.

However, the latest modified version of Chen's crash avoidance algorithm, approximately 11,000 lines of an assembler code that is small enough to fit in the available RAM of the DSP board, still contains bugs that are causing the program to halt.

5.2.3 Improvements

5.2.3.1 Sonar Sensors

One of the disadvantages of the current configuration of ROACH is the use of sonar sensors. Even though the sonar sensors can provide necessary information for crash avoidance of a 1:10 scale mobile robot, they do not apply for an actual size automobile because of their slow sampling rate. Even if the six sonar sets are sequentially fired to detect obstacles within 7ft without any interruptions, it takes at least 75.6 ms to complete the 360° scan because each sonar set requires 12.6 ms to collect the reading. If a vehicle is moving with a highway speed of 65 mph (95.3 ft/s), in the 75.6 ms, it can travel approximately 7.2 ft, more than the

distance the sonar sensors can detect. Thus, use of an alternative sensor that has faster sampling rate is suggested for implementation of Chen's crash avoidance system on an actual size automobile. The suggested sensors include radar, a vision sensor, and an infrared sensor.

5.2.3.2 DSP

As it is described in section 5.2.2, in order to reduce the program memory of Chen's crash avoidance system, floating-point numbers were changed to fixed-point numbers. However, since the fixed-point data, 16-bit, contains a half of the bits of the floating-point data, 32-bit, preciseness is lost.

According to the latest information found in the web site* of Analog Devices, a new DSP, ADSP-21060, can handle 32-bit floating-point processing, and it can store 128K words of 32-bit data, 256K words of 16-bit data, and 80K words of 48-bit instructions in its RAM. On the contrary, ADSP-2181, utilized in this thesis, can only process 16-bit fixed-point signals, and has RAM that can store only 16K words of 16-bit data and 16K words of 32-bit instructions. Thus, replacing the DSP with ADSP-21060 will ease the implementation of the crash avoidance system.

* http://www.analog.com

5.2.3.3 Bumper

Even though most of the collisions with static obstacles are head-on collisions, a moving obstacle may run into the side or rear of the robot, where the front bumper cannot protect the components. Thus, installation of a bumper which surrounds the robot 360° is suggested to protect the robot from unexpected side-collisions and rear-collisions.

5.2.3.4 Alarm Set

As mentioned in section 2.3.6.3, FL1, one of the flag output pins of the DSP board, can be used as an alarm set. Since the executable program including Chen's crash avoidance system can modulate the output of FL1, it may be used as a warning signal for prediction of up-coming collisions by connecting it to a buzzer or a flashing light bulb. This feature will allow an observer to understand when the crash avoidance system takes over the operator's control of the robot.

5.2.3.5 Feedback Devices

The current configuration of ROACH is an open loop. In other words, the input values of the speed and steering angle, fed into the crash avoidance system, are not direct measurements from the actuators. The values are taken directly from the inputs of the transmitter. Thus, in order to provide feedback to the configuration, installation of a tachometer and a potentiometer for the motor and servo, respectively, are suggested. However, since all programmable flag pins are

currently in use, an additional circuit has to be built for the DSP board to receive the signals from the feedback devices.

5.3 Future Work

Primary future work includes modifying Chen's crash avoidance system, so that the algorithm can be actually tested in ROACH. The robot may require some changes after Chen's crash avoidance system is implemented. For example, one of the unknown factors that could slow down the performance of ROACH is the total execution time of Chen's crash avoidance system. Even though, in the current configuration, a 12.6 ms execution time is reserved for the crash avoidance algorithm, the actual time required by Chen's crash avoidance system will not be known until it is actually implemented. If the algorithm takes more than 12.6 ms, it may have to be divided into parts, so that a portion of the algorithm can be executed at each 12.6 ms. This modification is expected to slow down the performance of the robot since the control algorithm takes more time to update output pulses to the servo and speed controller.

Implementation of Chen's crash avoidance system into ROACH should definitely provide insights into its implementation on an actual size automobile. In other words, the result of the implementation of Chen's crash avoidance system into ROACH should become a key to a crash avoidance of future automobiles. I certainly hope that this technology will help to reduce the tragedies of automobile crashes in the future.

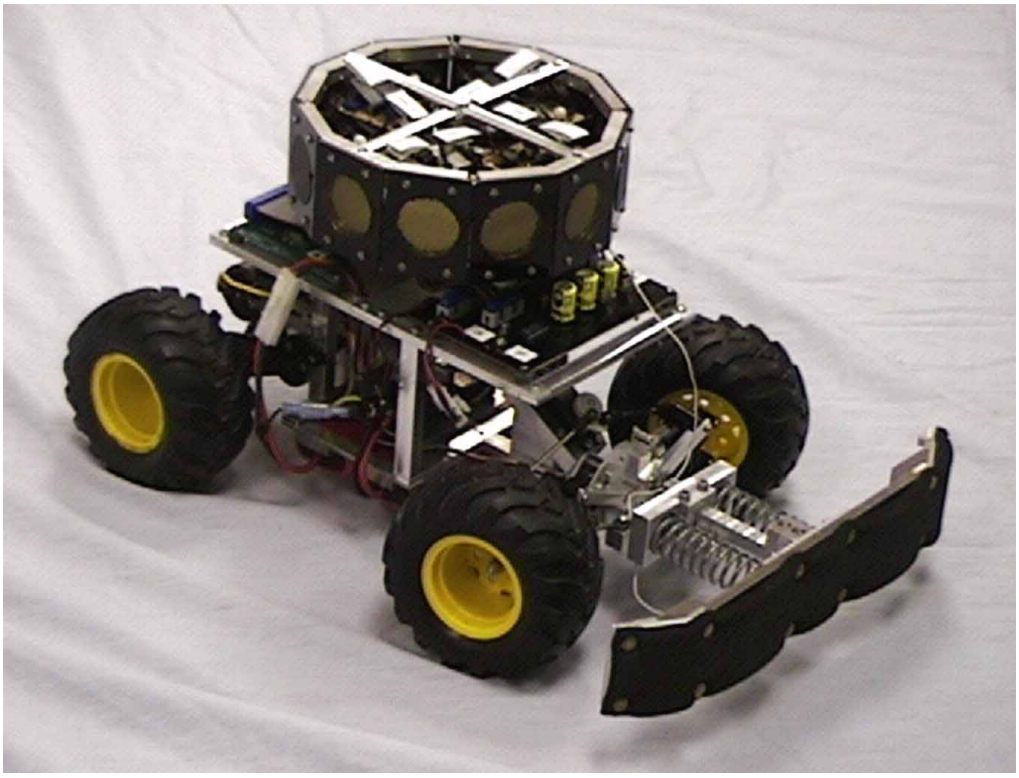

Fig. 1.1 ROACH - Picture

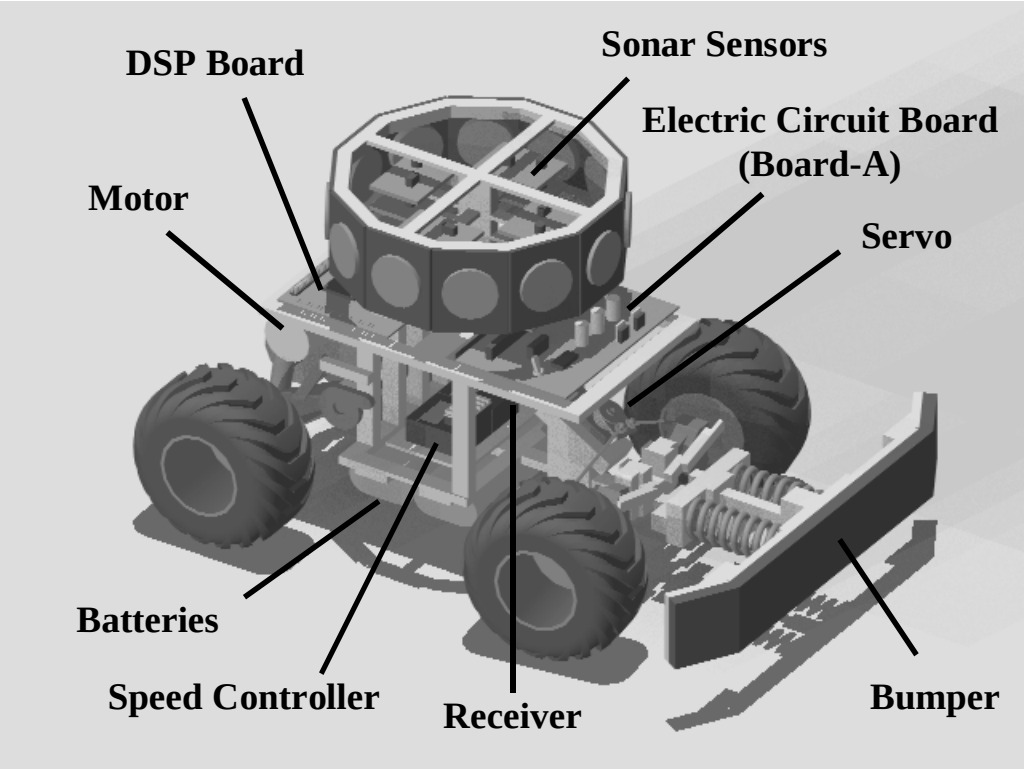

Fig. 2.1 ROACH – AutoCAD Drawing

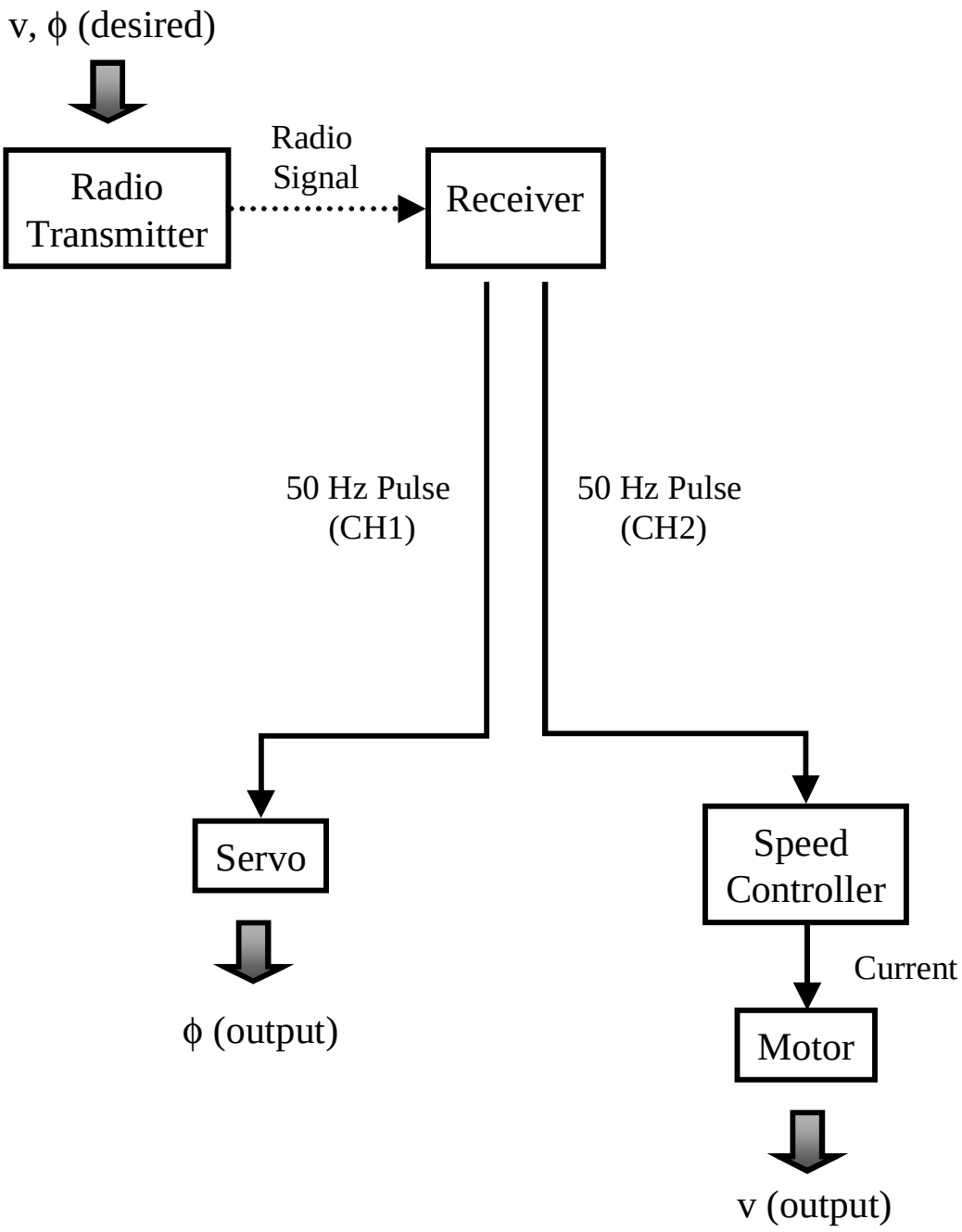

Fig. 2.2 Information Process – RC Car

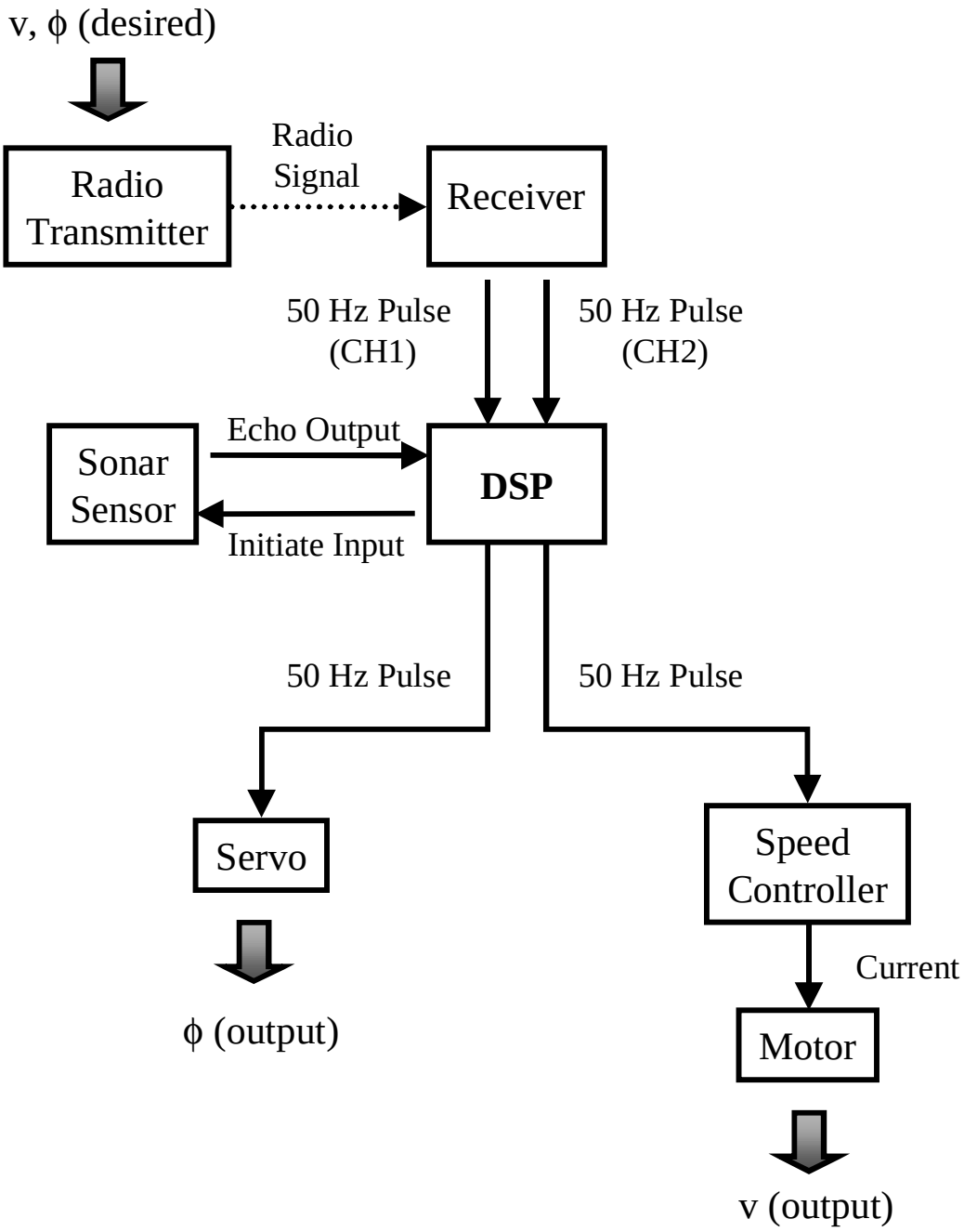

Fig. 2.3 Information Process - ROACH

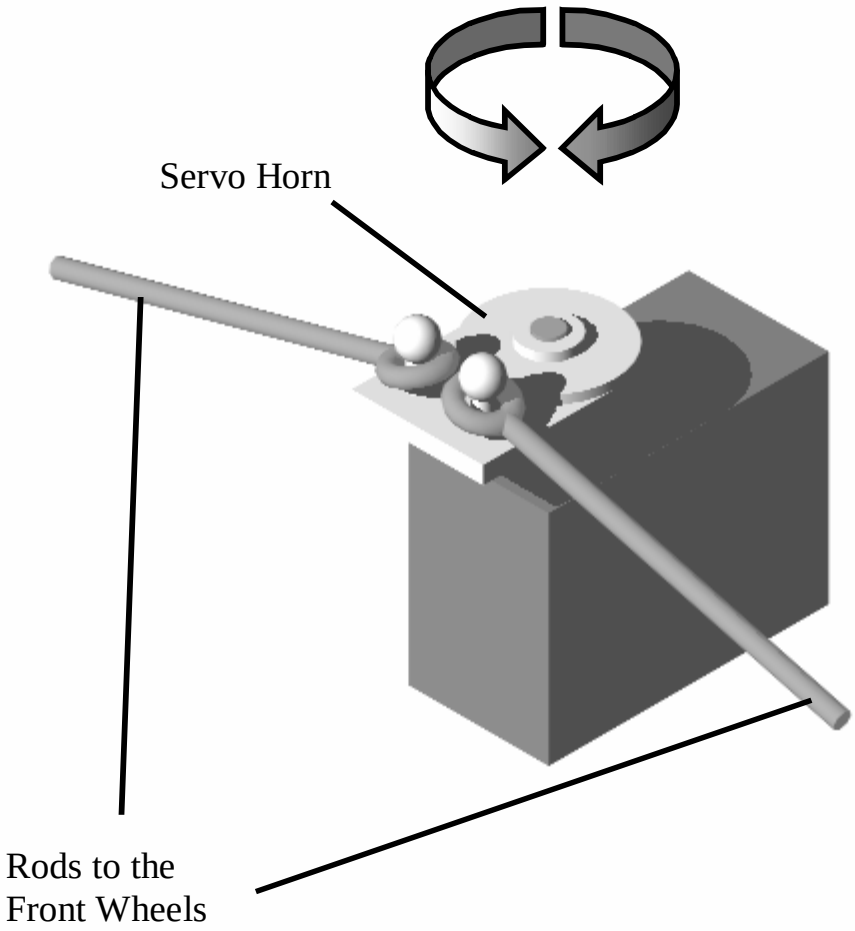

Fig. 2.4 Servo

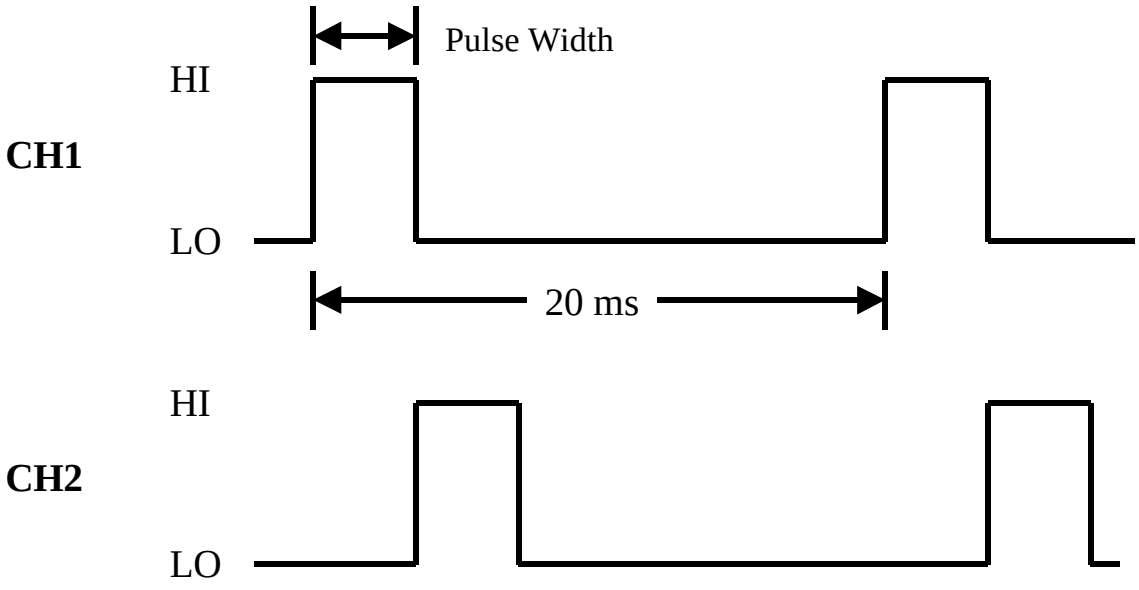

Fig. 2.5 50 Hz Pulses from the Receiver

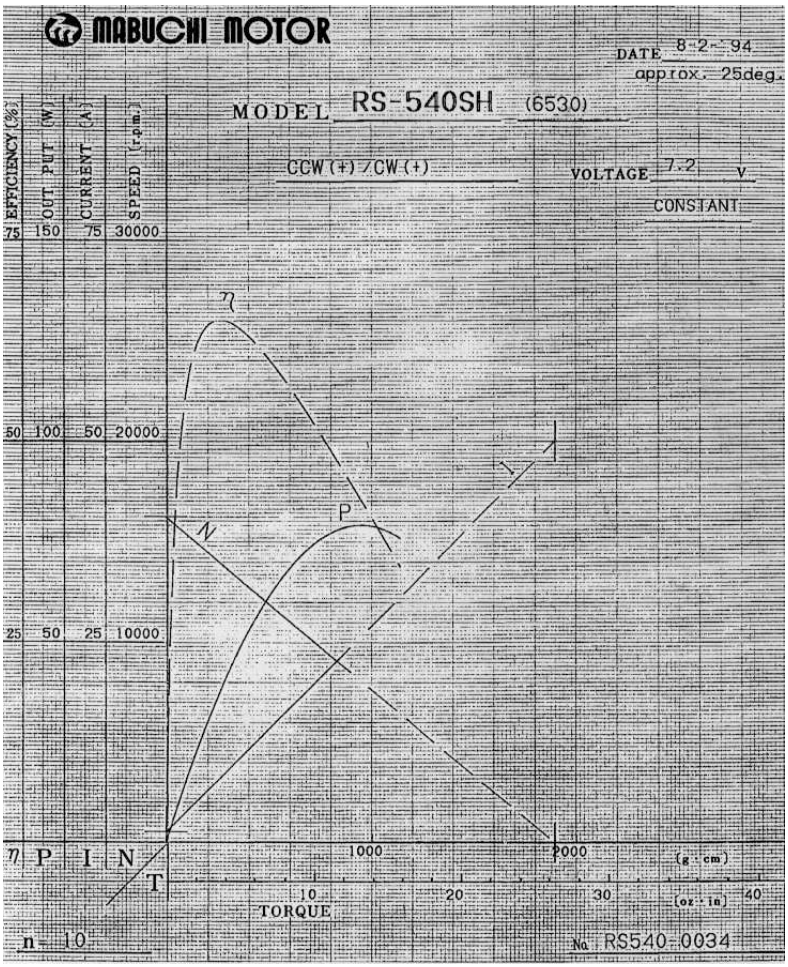

Fig. 2.6 Mabuchi Motor RS-540SH Data Sheet

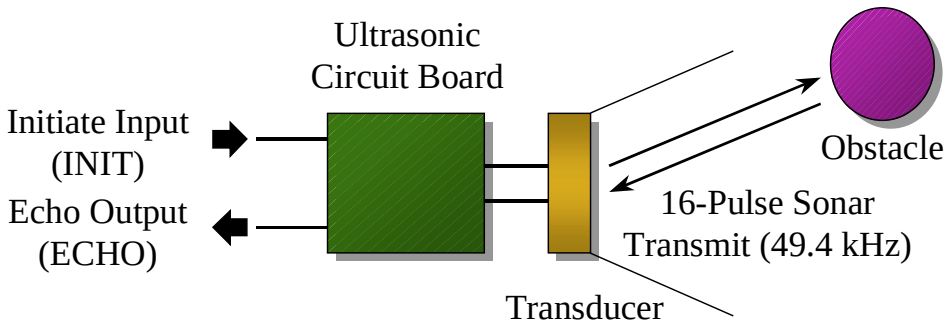

Fig. 2.7 Sonar Sensor Signals

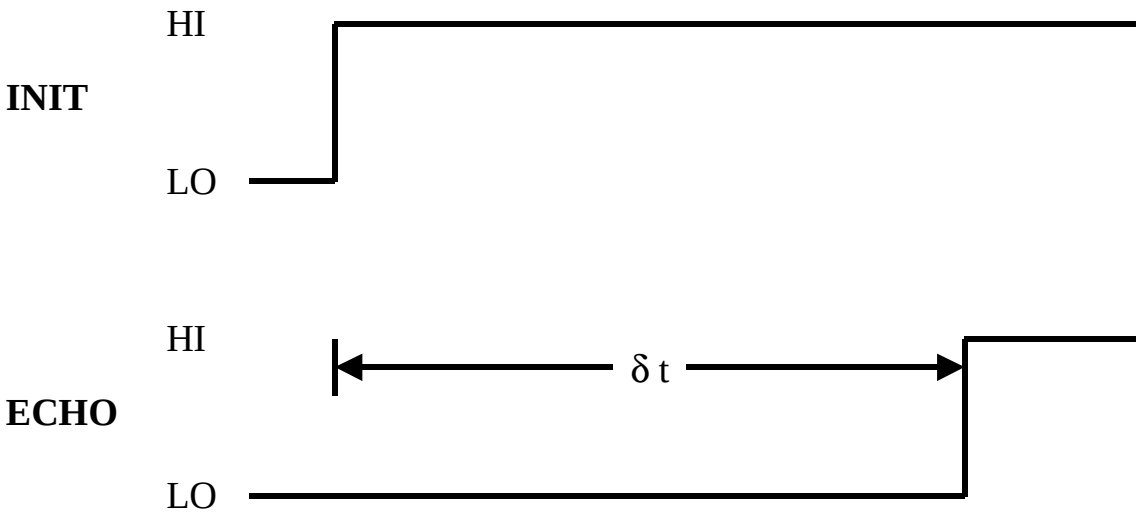

Fig. 2.8 Time-of-Flight

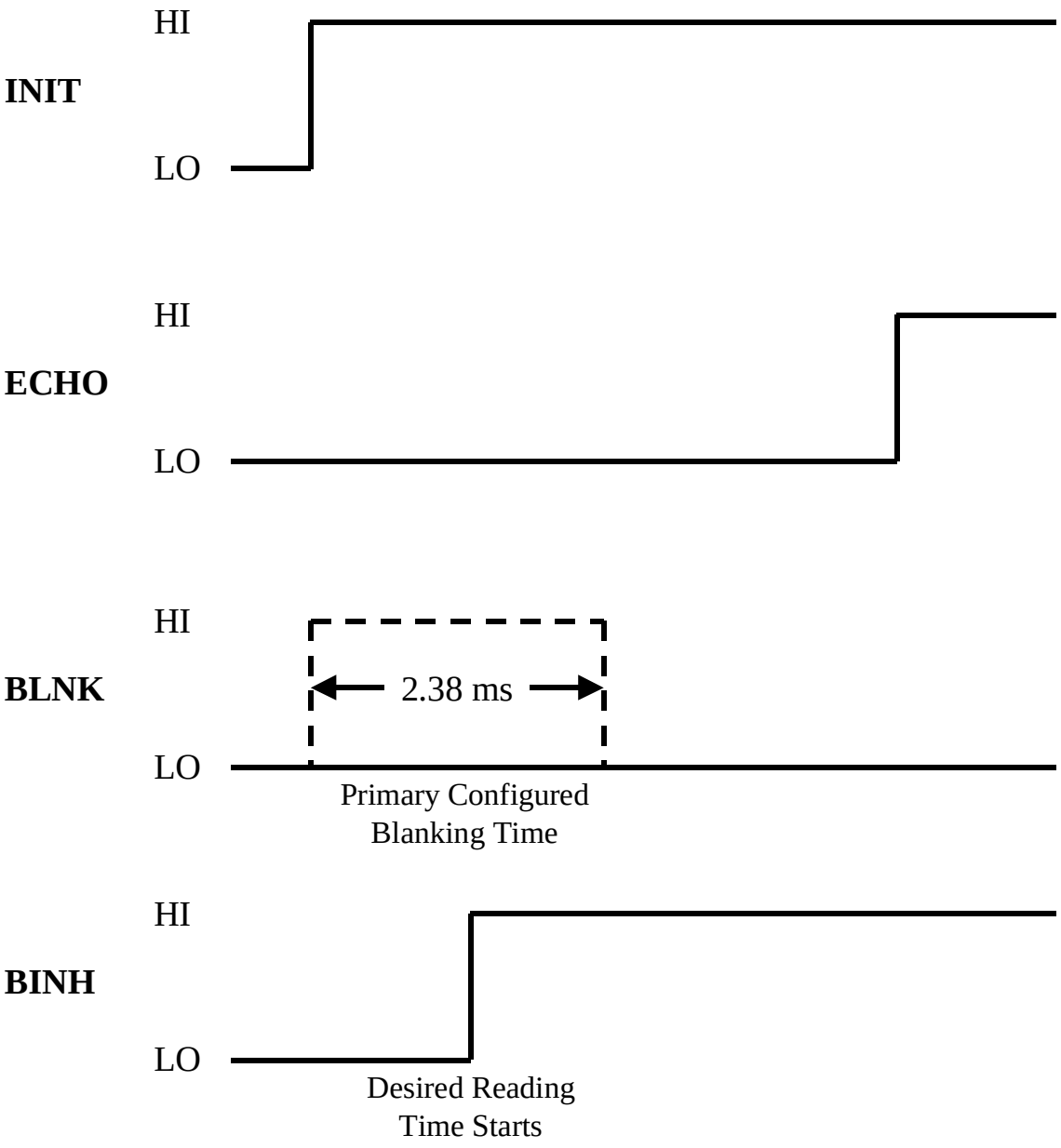

Fig. 2.9 Blanking Time Configuration

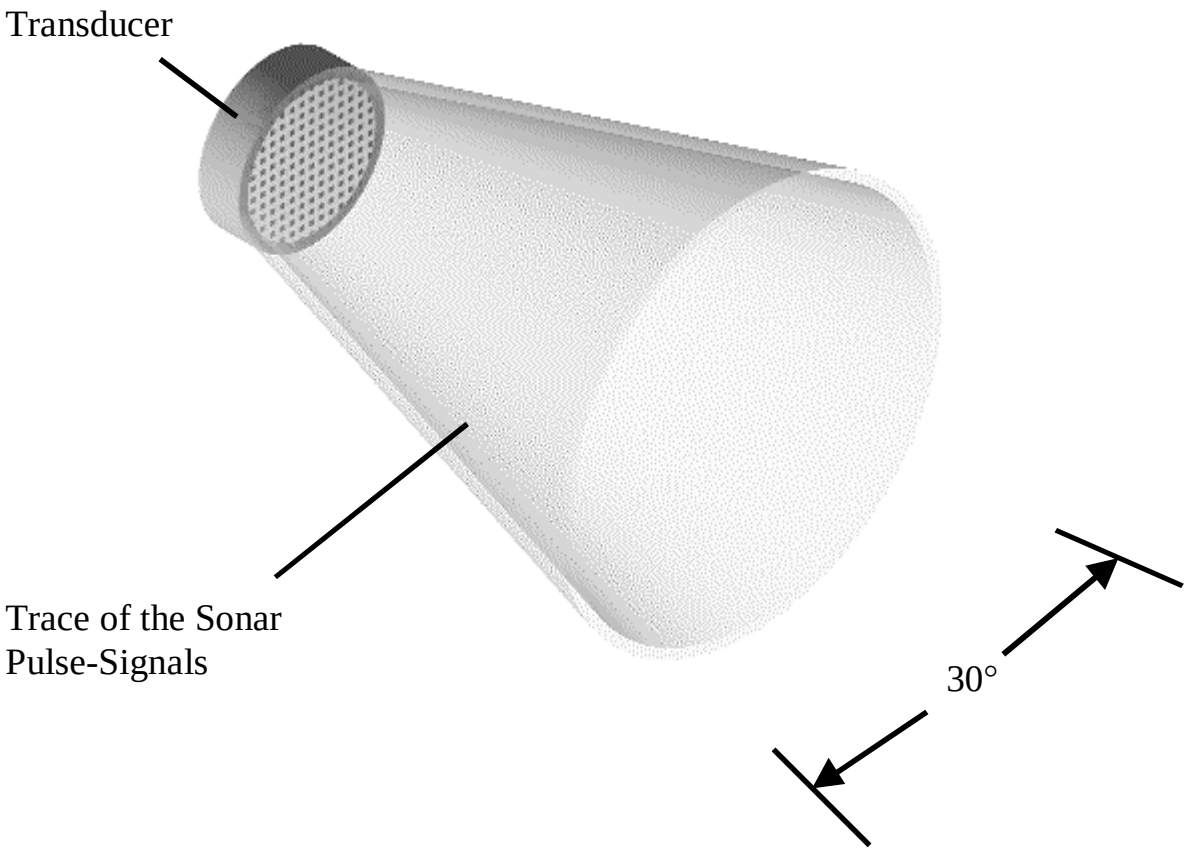

Fig. 2.10 Trace of the Sonar Pulse-Signals

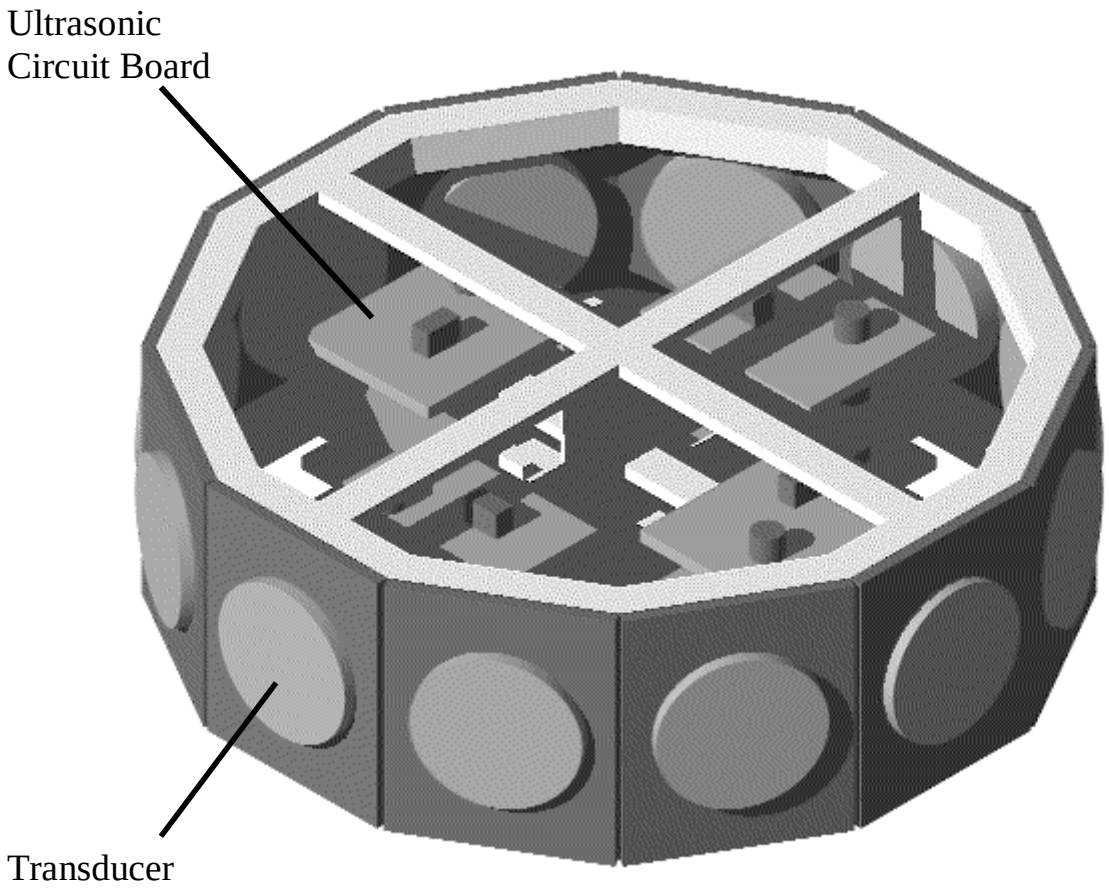

Fig. 2.11 Circular Array of the Sonar Sensors

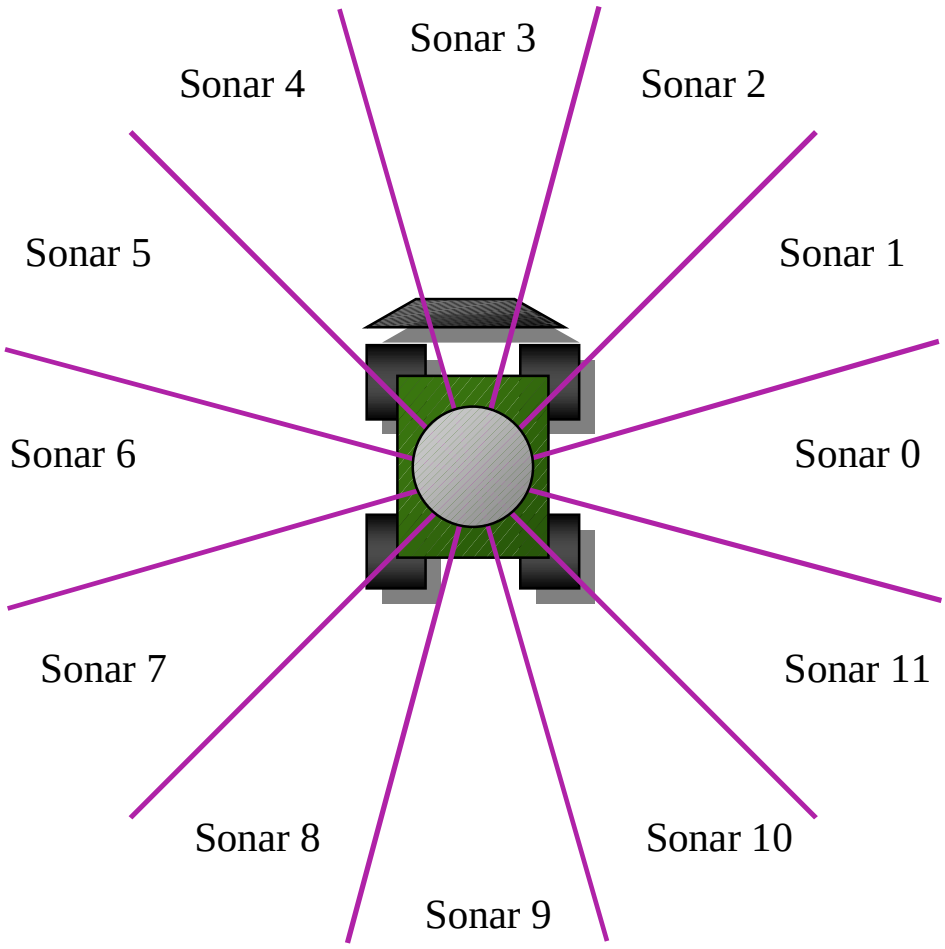

Fig. 2.12 Numbers of the Sonar Sensors

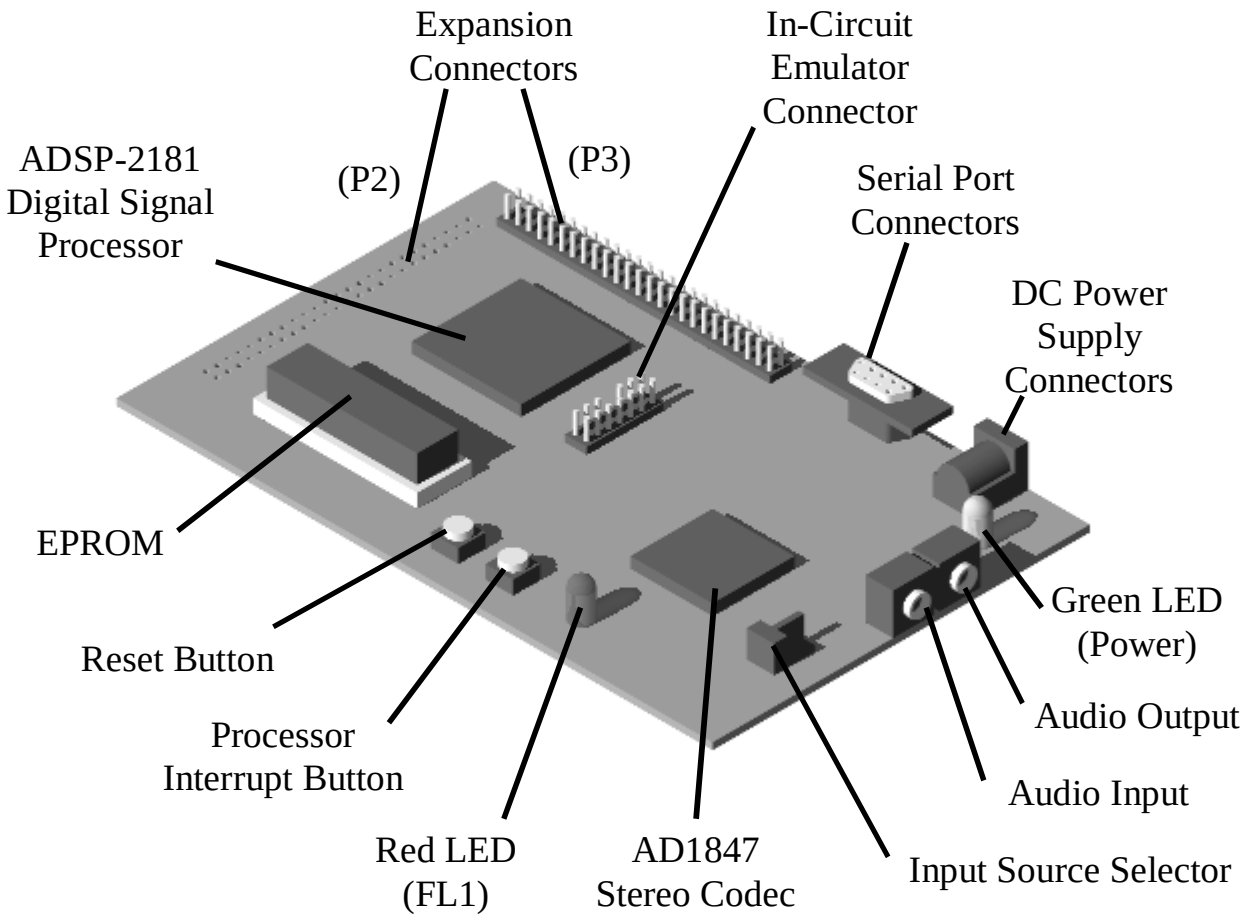

Fig. 2.13 ADSP-2181 EZ-KIT Lite

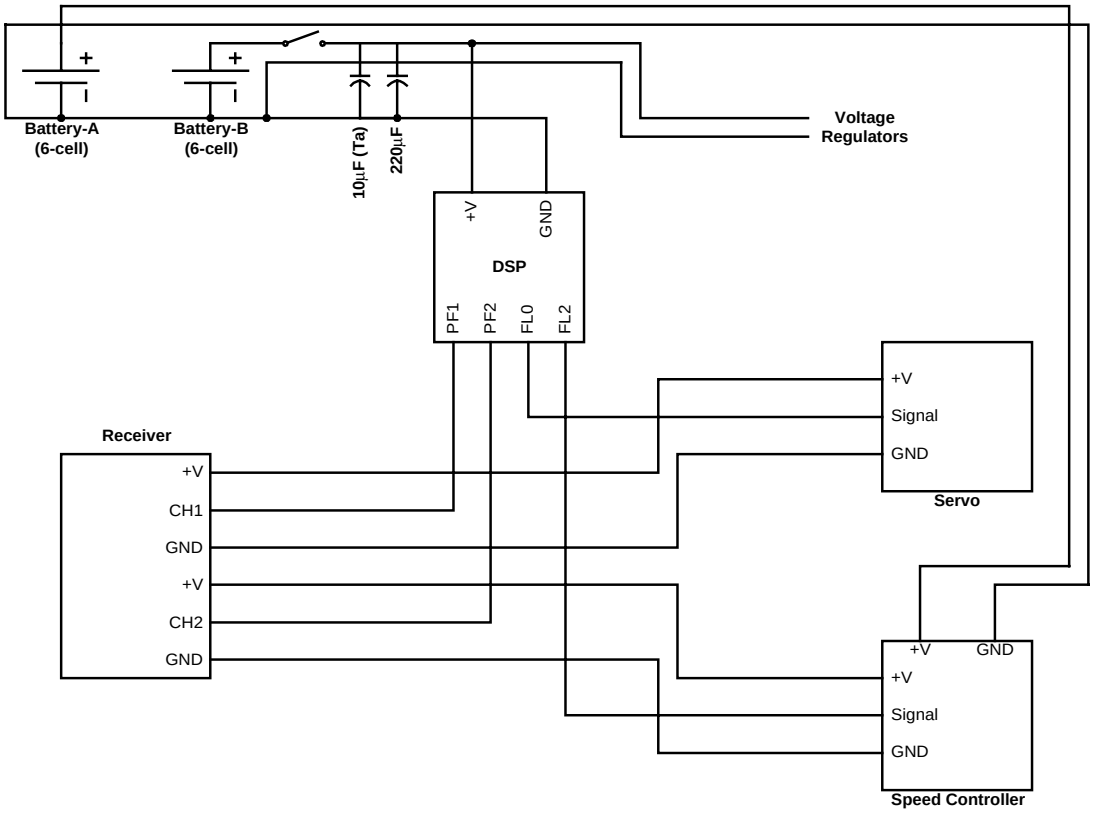

Fig. 2.14 Electric Circuit – the DSP Board to Other Components

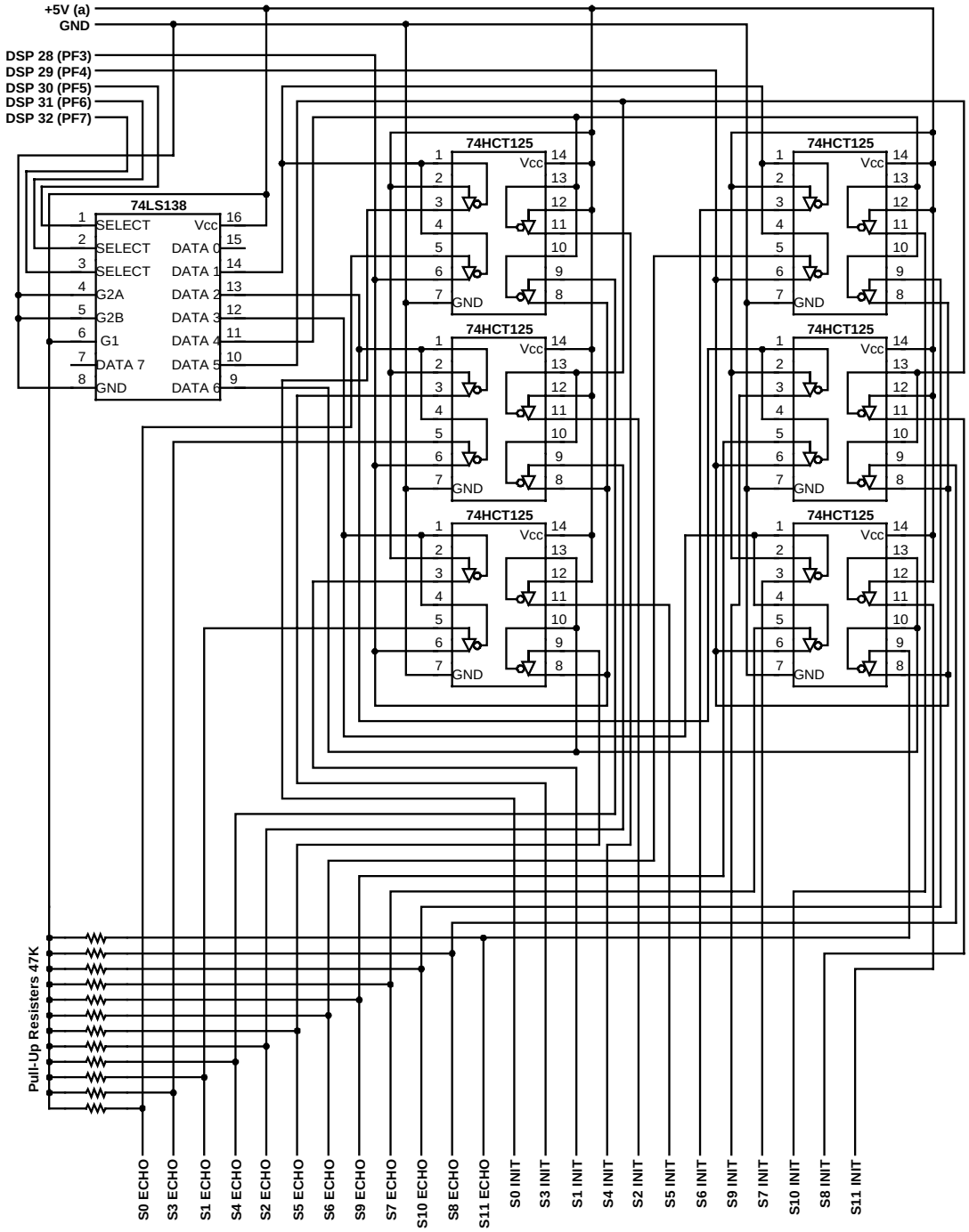

Fig. 2.15 Sonar Interface Circuit

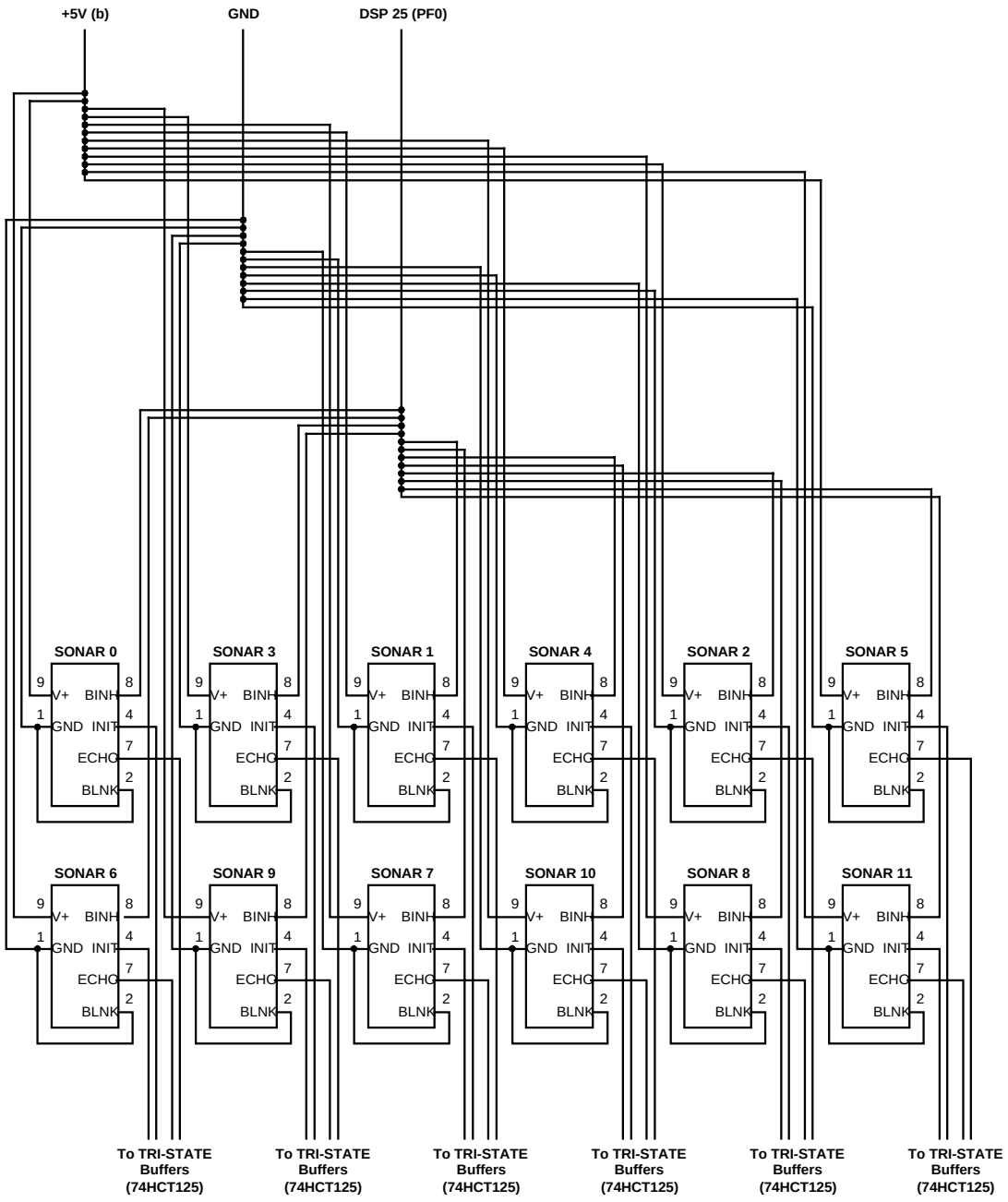

Fig. 2.16 Sonar Sensors Wiring

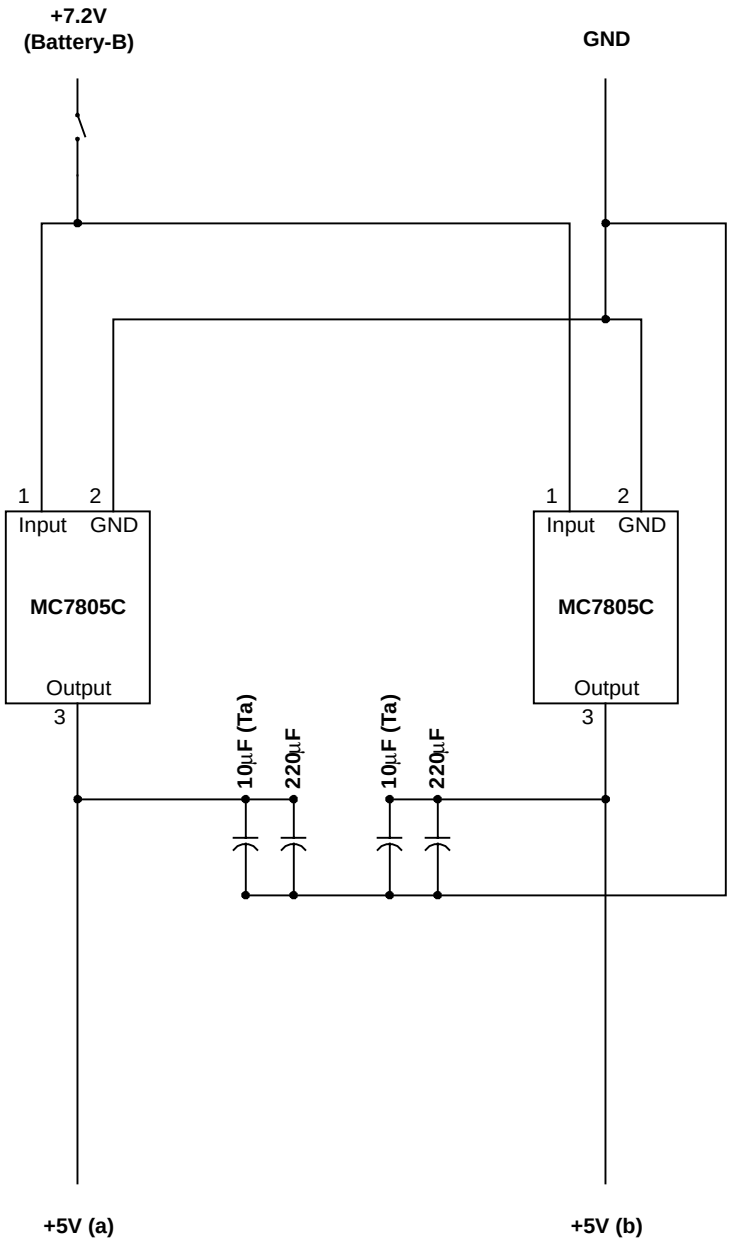

Fig. 2.17 Voltage Regulators

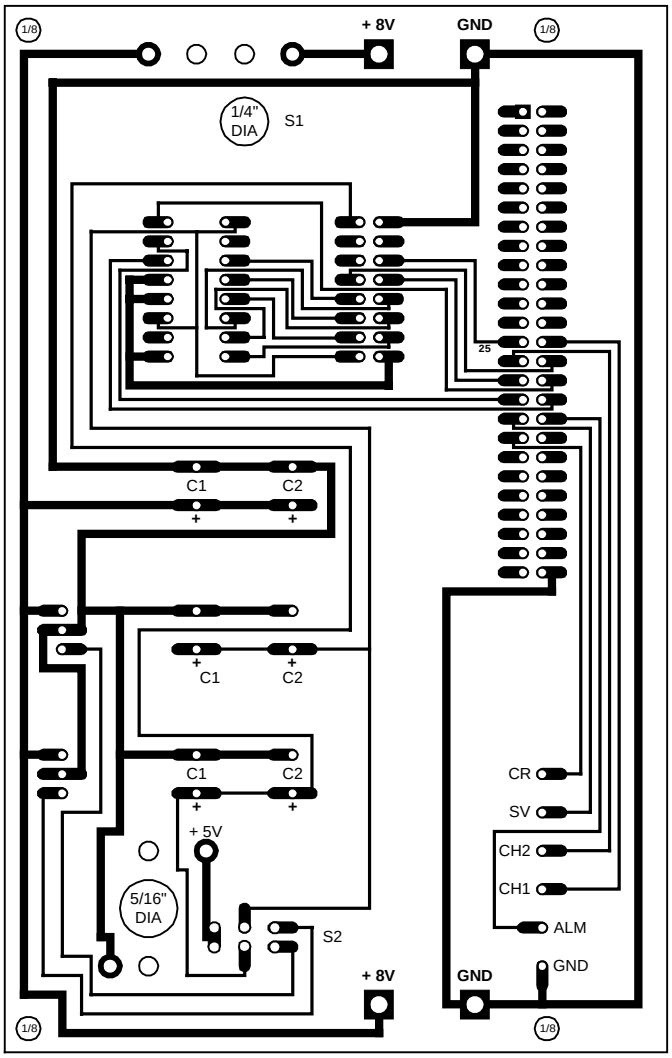

Fig. 2.18 Electric Circuit Board Layout – Board-A

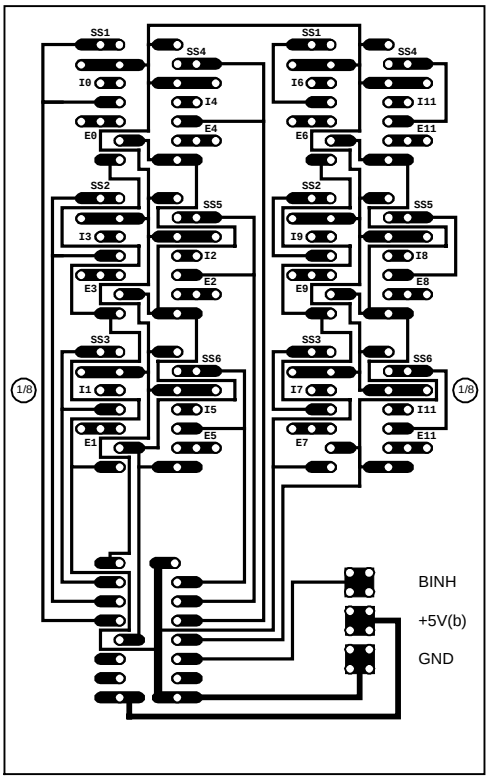

Fig. 2.19 Electric Circuit Board – Board-B

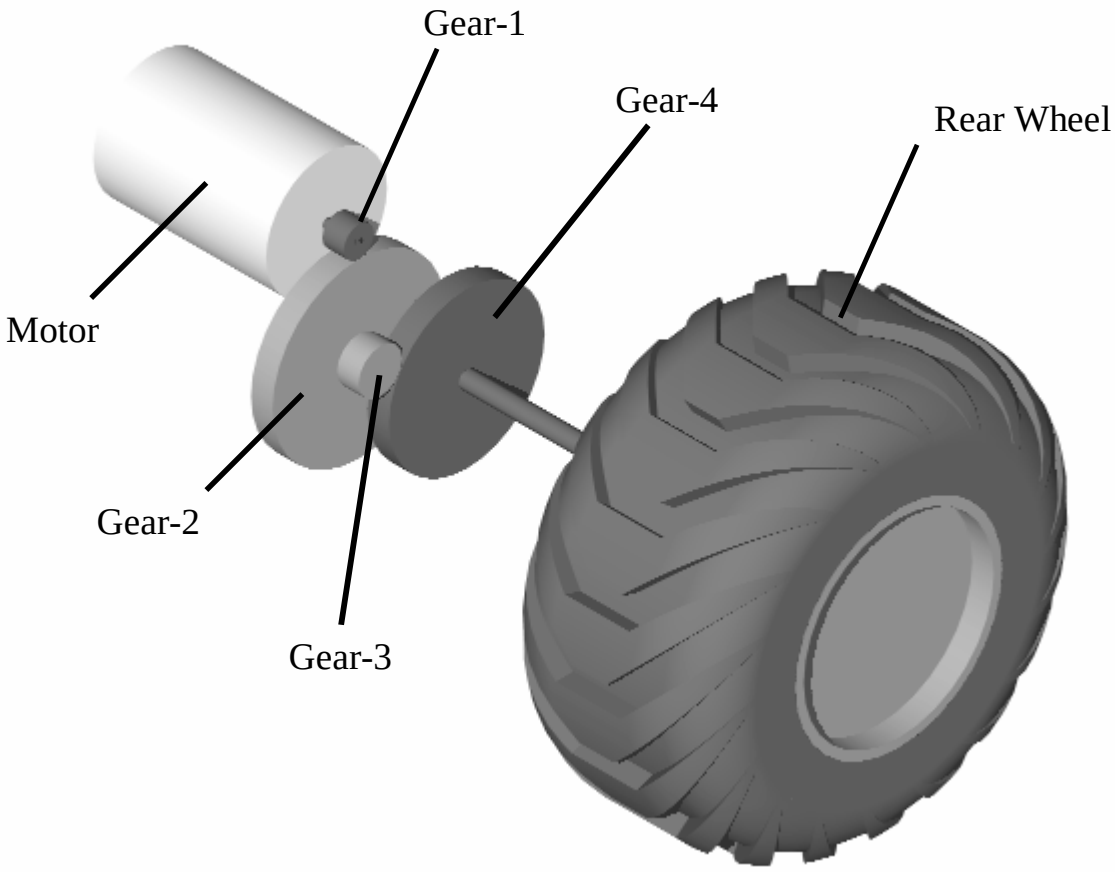

Fig. 2.20 Rear Wheel Component

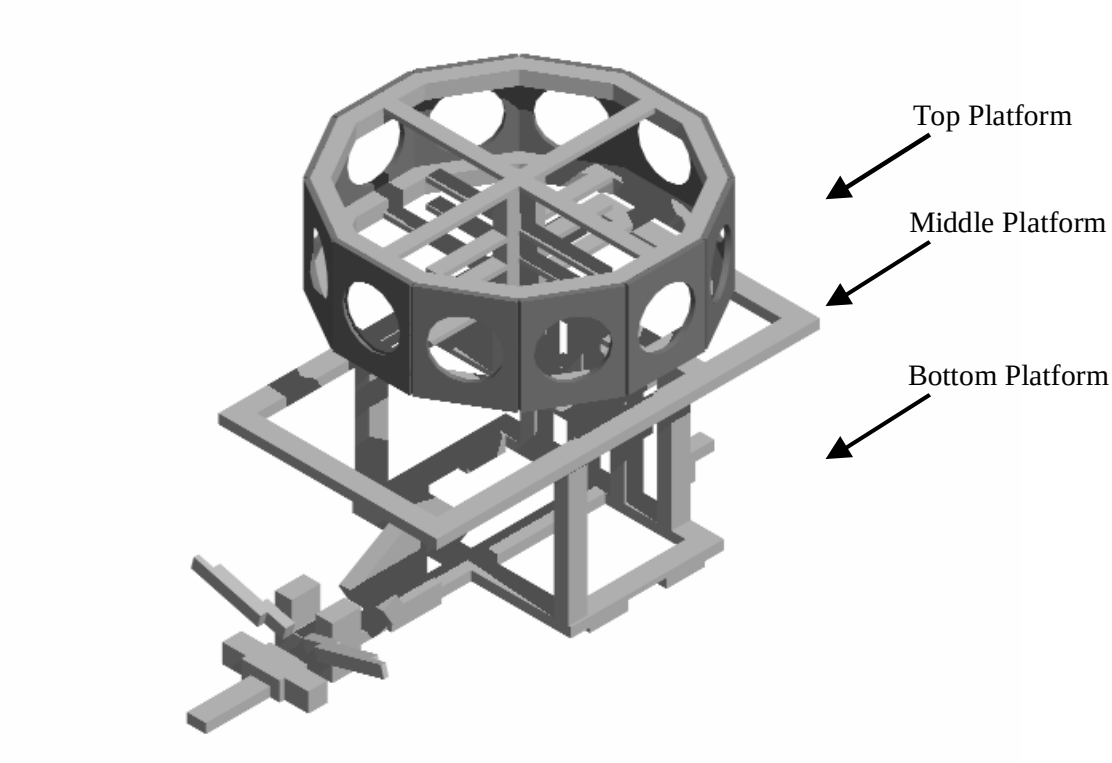

Fig. 2.21 Frame

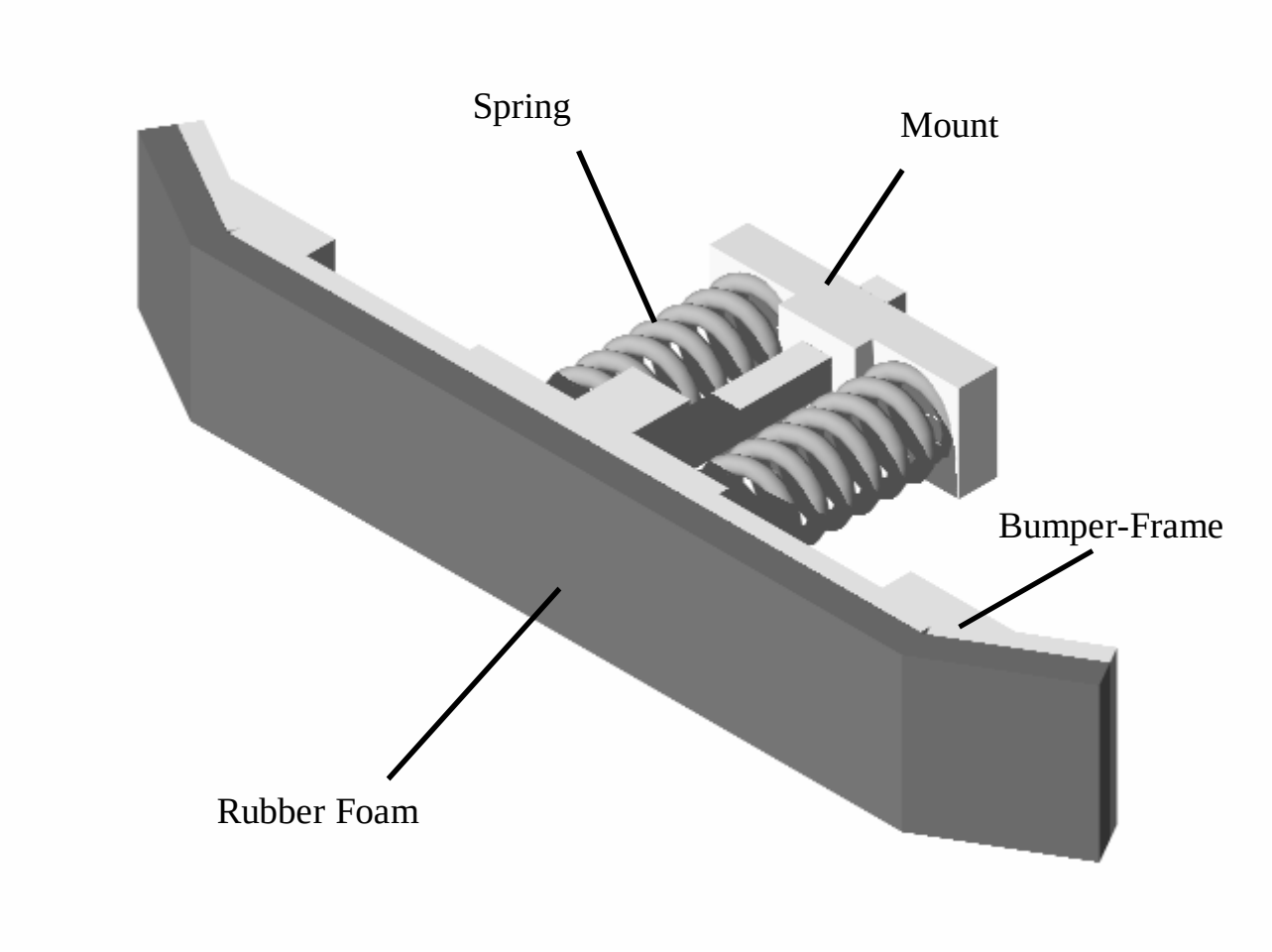

Fig. 2.22 Bumper

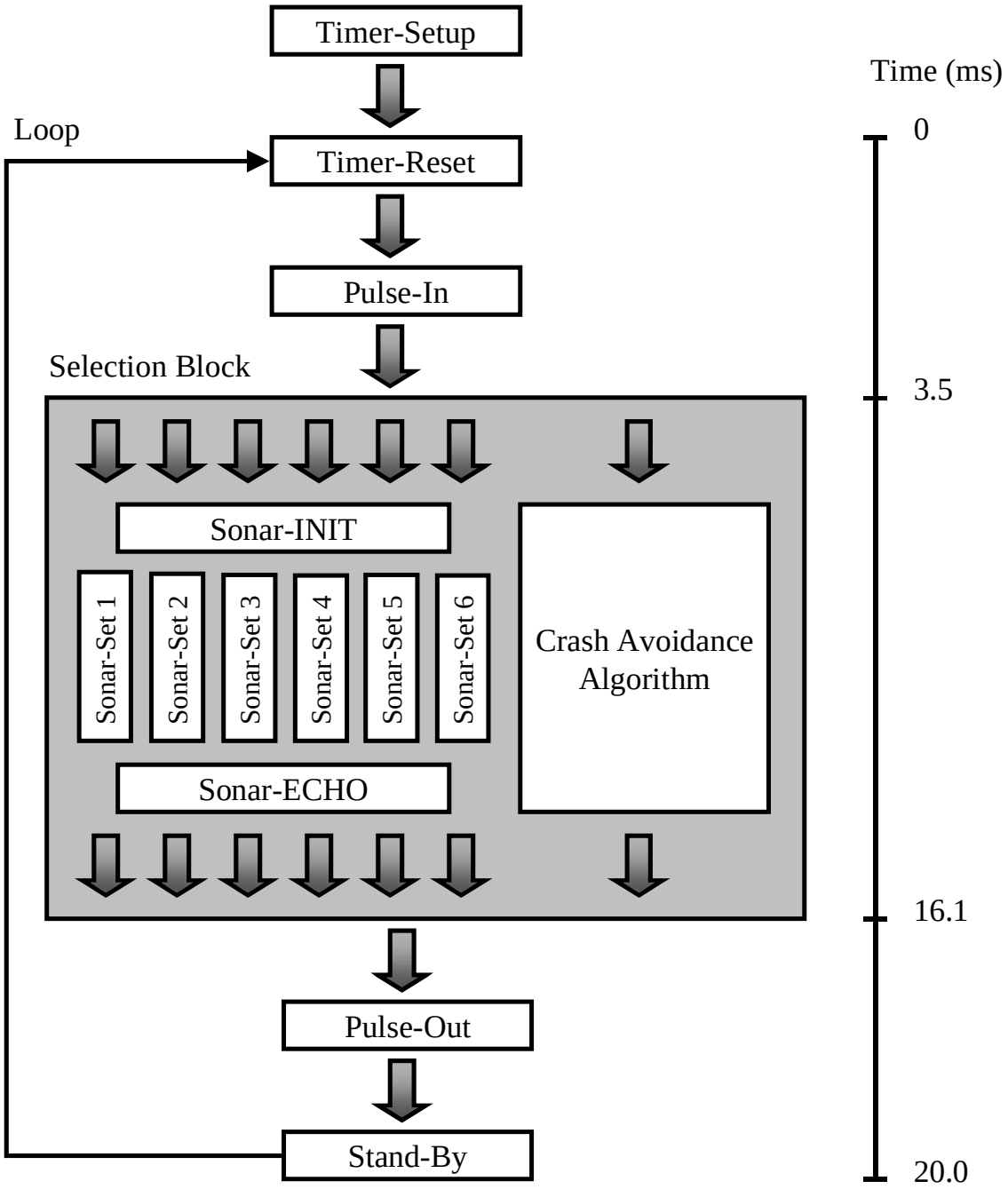

Fig. 3.1 Time Schedule for the Control Algorithm

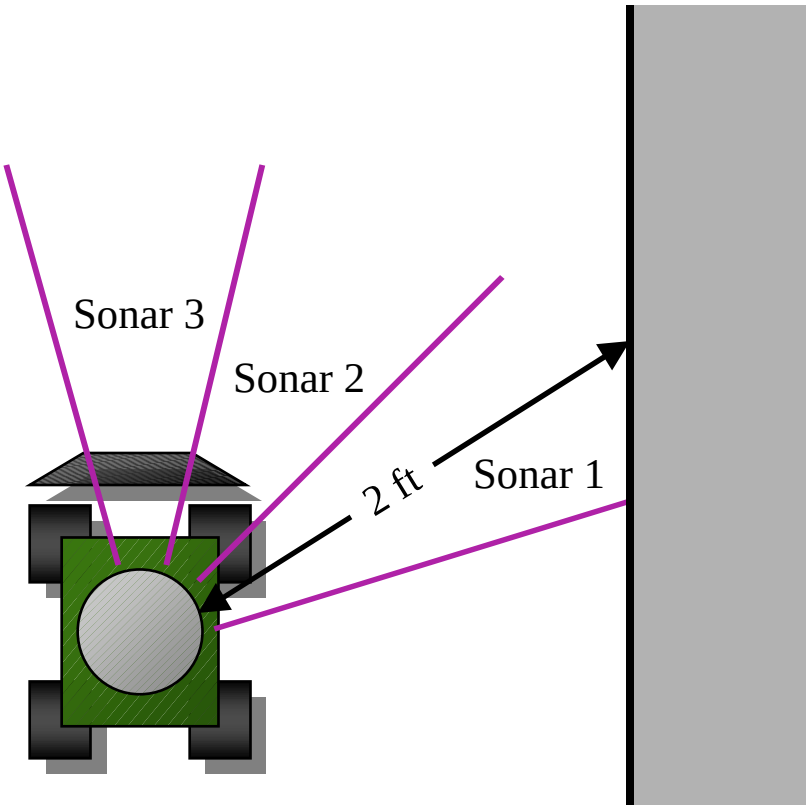

Fig. 3.2 Single Wall Following

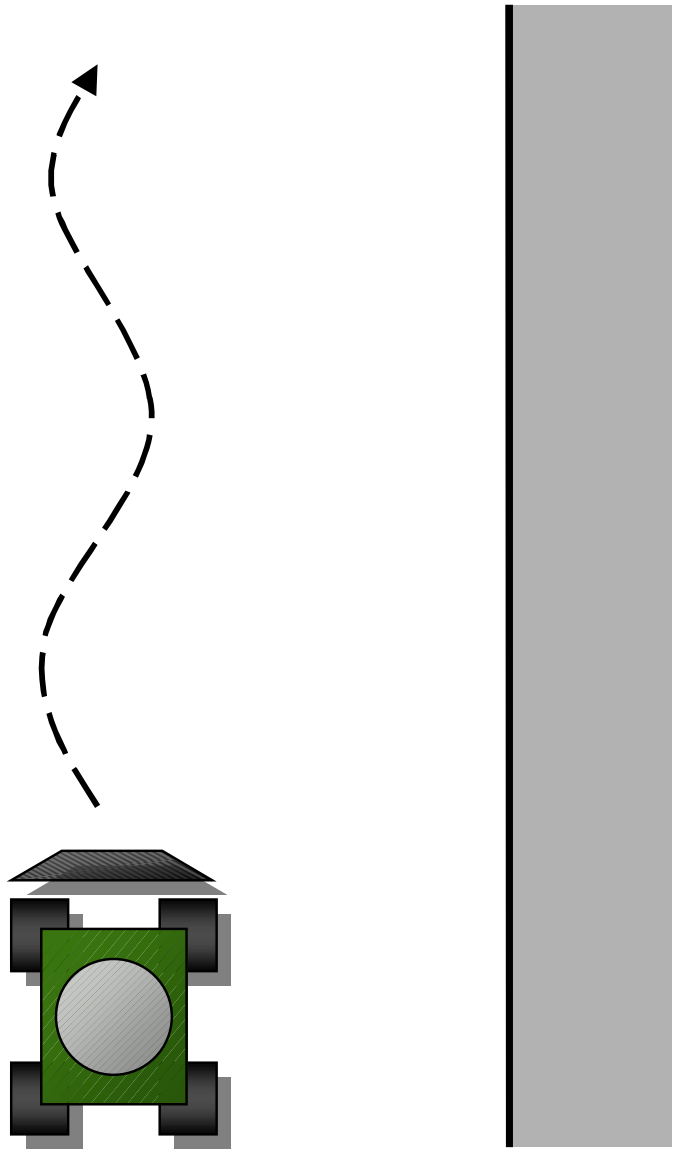

Fig. 4.1 Wall Following – Straight

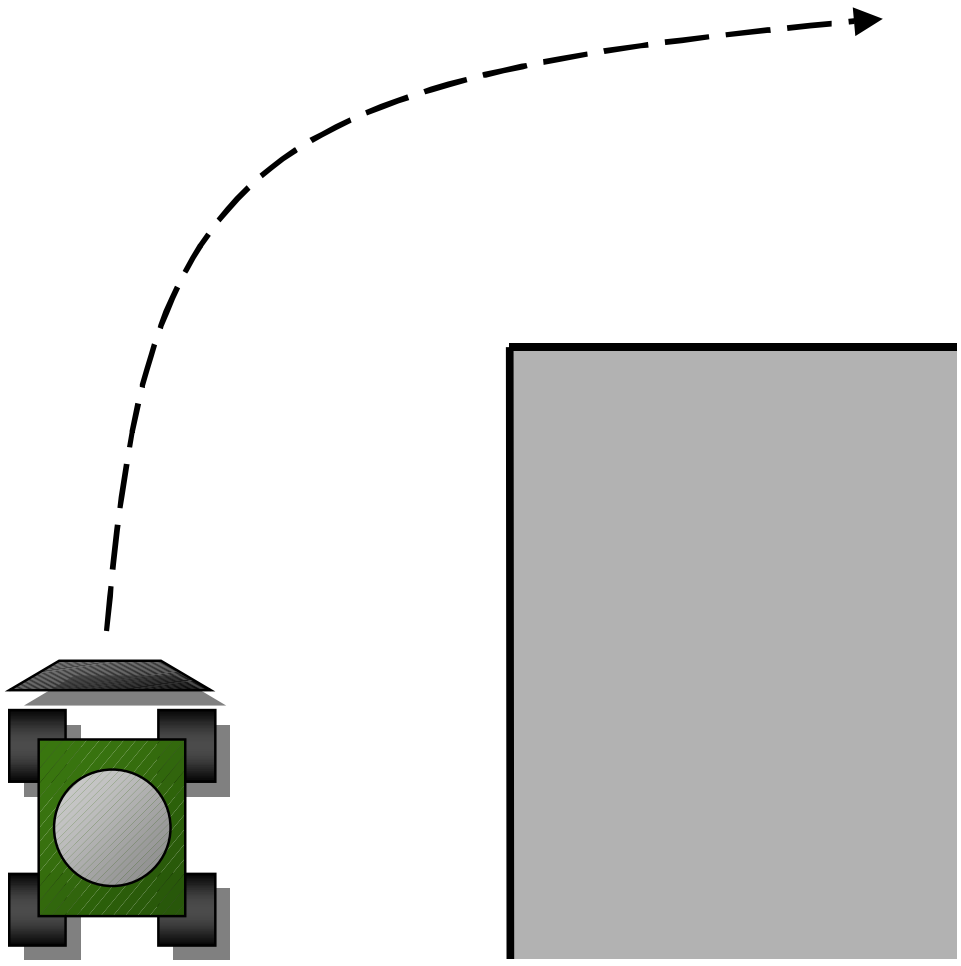

Fig. 4.2 Wall Following – Right Turn

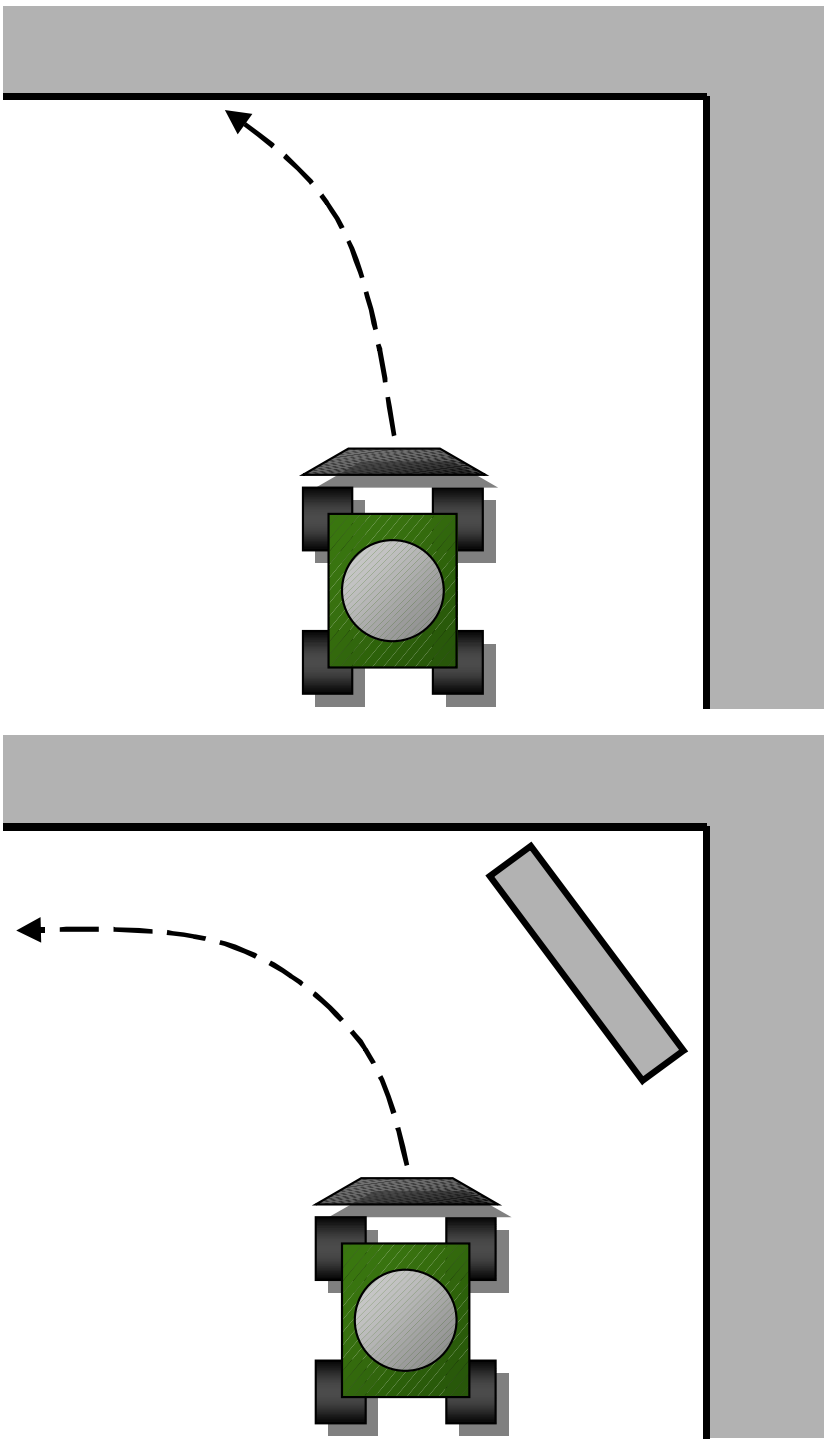

Fig. 4.3 Wall Following – Left Turn

Pin Number	Signal Name	Pin Number	Signal Name
1	A0	2	A1
3	A2	4	A3
5	A4	6	A5
7	A6	8	A7
9	A8	10	A9
11	A10	12	A11
13	A12	14	A13
15	D0	16	D1
17	D2	18	D3
19	D4	20	D5
21	D6	22	D7
23	D8	24	D9
25	D10	26	D11
27	D12	28	D13
29	D14	30	D15
31	D16	32	D17
33	D18	34	D19
35	D20	36	D21
37	D22	38	D23
39	WR	40	RD
41	IOMS	42	BMS
43	DMS	44	CMS
45	PMS	46	BR
47	BGH	48	BG
49	VCC	50	GND

Table 2.1 Expansion Connector – P2

Pin Number	Signal Name	Pin Number	Signal Name
1	GND	2	IAD0
3	IAD1	4	IAD2
5	IAD3	6	IAD4
7	IAD5	8	IAD6
9	IAD7	10	IAD8
11	IAD9	12	IAD10
13	IAD11	14	IAD12
15	IAD13	16	IAD14
17	IAD15	18	GND
19	IACK	20	IAL
21	IS	22	IWR
23	IRD	24	GND
25	PF0	26	PF1
27	PF2	28	PF3
29	PF4	30	PF5
31	PF6	32	PF7
33	FL0	34	FL1
35	FL2	36	CLKOUT
37	RESET	38	IRQL0
39	IRQL1	40	IRQ2
41	PWD	42	PWDACK
43	CODECDIS	44	TXD0
45	TFS0	46	RFS0
47	RXD0	48	SCK0
49	VCC	50	GND

Table 2.2 Expansion Connector – P3

INPUT			OUTPUT	
Pin 1 (PF5)	Pin 2 (PF6)	Pin 3 (PF7)	Selected Output	Sonar Set
0	0	0	Pin 15	none
1	0	0	Pin 14	Sonar Set 1
0	1	0	Pin 13	Sonar Set 2
1	1	0	Pin 12	Sonar Set 3
0	0	1	Pin 11	Sonar Set 4
1	0	1	Pin 10	Sonar Set 5
0	1	1	Pin 9	Sonar Set 6
1	1	1	Pin 7	none

Table 2.3 Demultiplexer

```
jump start; rti; rti; rti;                {00: reset }
rti; rti; rti; rti;                       {04: IRQ2 }
rti; rti; rti; rti;                       {08: IRQL1}
rti; rti; rti; rti;                       {0c: IRQL0 }
rti; rti; rti; rti;                       {10: SPORT0 tx }
rti; rti; rti; rti;                       {14: SPORT0 rx }
rti; rti; rti; rti;                       {18: IRQE }
rti; rti; rti; rti;                       {1c: BDMA }
rti; rti; rti; rti;                       {20: SPORT1tx or IRQ1}
rti; rti; rti; rti;                       {24: SPORT1rx or IRQ0}
jump update_signal; rti; rti; rti;        {28: timer }
rti; rti; rti; rti;                       {2c: power down }
```

Table 3.1 Interrupt Vector Table for the Control Algorithm

REFERENCES

ADSP-2100 Family Assembler Tools & Simulator Manual, Analog Devices, Inc., One Technology Way, Norwood, MA 02062, 1994.

ADSP-2100 Family C Tools Manual, Analog Devices, Inc., One Technology Way, Norwood, MA 02062, 1994.

ADSP-2100 Family EZ-KIT Lite Reference Manual, Analog Devices, Inc., One Technology Way, Norwood, MA 02062, 1995.

ADSP-2100 Family User's Manual, Analog Devices, Inc., One Technology Way, Norwood, MA 02062, 1995.

Polaroid 6500 Series Sonar Ranging Module, Polaroid Corporation, 1 Upland Road, Norwood, MA 02062, 1991.

Power Pulse Speed Control, Dynamite, c/o Horizontal Hobby Distributors, Inc., 4105 Fieldstone Road, Champaign, IL 61821, 1996.

Traffic Safety Facts 1996 Overview, National Highway Traffic Safety Administration (NHTSA), U.S. Department of Transportation, 400 Seventh Street, S.W., Washington, D.C., 1996.

Arkin, R. C., "Motor Schema-Based Mobile Robot Navigation," *The International Journal of Robotics Research*, vol. 8, pp. 92-112, 1989.

Beer, R. D. and Chiel, H. J., "Simulations of Cockroach Locomotion and Escape," *Biological Neural Networks in Invertebrate Neuroethology and Robotics*, Beer, R. D., Ritzmann, R. E. and McKenna, T. ed., Academic Press, chap. XII, 1993.

Borenstein, J. and Koren, Y., "Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments," *Proceeding of the IEEE International Conference on Robotics and Automation*, pp. 572-577, 1990.

Brooks, R. A., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, pp. 14-23, 1986.

Chen, C-T., "A Crash Avoidance System Based Upon the Cockroach Escape Response," Ph.D. dissertation, Case Western Reserve University, Department of Mechanical and Aerospace Engineering, January 1996.

Chen, C-T., Quinn, R. D. and Ritzmann, R. E., "A Crash Avoidance System Based Upon the Cockroach Escape Response Circuit," *Proceeding of the IEEE International Conference on Robotics and Automation*, pp. 2007-2012, 1997.

Gourley, C. and Trivedi, M., "Sonar Based Obstacle Avoidance and Mapping for Fast Mobile Robots," *Proceeding of the IEEE International Conference on Robotics and Automation*, pp. 1306-1311, 1994.

Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, vol. 5, pp. 90-98, 1986.

Koren, Y. and Borenstein, J., "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation," *Proceeding of the IEEE International Conference on Robotics and Automation*, pp. 1398-1404, 1991.

Martens, J. H., "Enhanced Teleoperation of a Mobile Robot," M.S. thesis, Case Western Reserve University, Department of Electrical Engineering and Applied Physics, May 1993.

Ming, L., Zalin, G. and Shuzi, Y., "Mobile Robot Fuzzy Control Optimization Using Genetic Algorithm," *Artificial Intelligence in Engineering*, vol. 10, pp. 293-298, 1996.

Newman, W. S. and Hogan, N., "High Speed Robot Control and Obstacle Avoidance Using Dynamic Potential Functions," *Proceeding of the IEEE International Conference on Robotics and Automation*, pp. 12-24, 1987.

Pao, Y. H. "Adaptive Pattern Recognition and Neural Networks," Adison-Wesley Publishing Company, Inc., 1989.

Pomerleau, D. A., "Neural Network Perception for Mobile Robot Guidance," Kluwer Academic Publishers, 1993.

Ritzmann, R. E., "The Neural Organization of Cockroach Escape and Its Role in Context-Dependent Orientation," *Biological Neural Networks in Invertebrate Neuroethology and Robotics*, Beer, R. D., Ritzmann, R. E. and McKenna, T. ed., Academic Press, chap. VI, 1993.

Schiller, I. and Tench, K.A., "A Neural-Network-Based Autonomous Underwater Vehicle Guidance System," *Proceedings of the 6th International*

Symposium on Unmanned Untethered Submersible Technology, pp. 312-319, 1989.

Touretzky, D. S. and Pomerleau, D. A. "What's Hidden in the Hidden Layers?" *Byte*, pp. 227-233, 1989.

Yung, N. H. C. and Ye, C., "Self-Learning Fuzzy Navigation of Mobile Vehicle," *Proceedings of the 1996 3rd International Conference on Signal Processing*, vol. 2, pp. 1465-1468, 1996.

APPENDIX A

ROACH DIMENSIONS

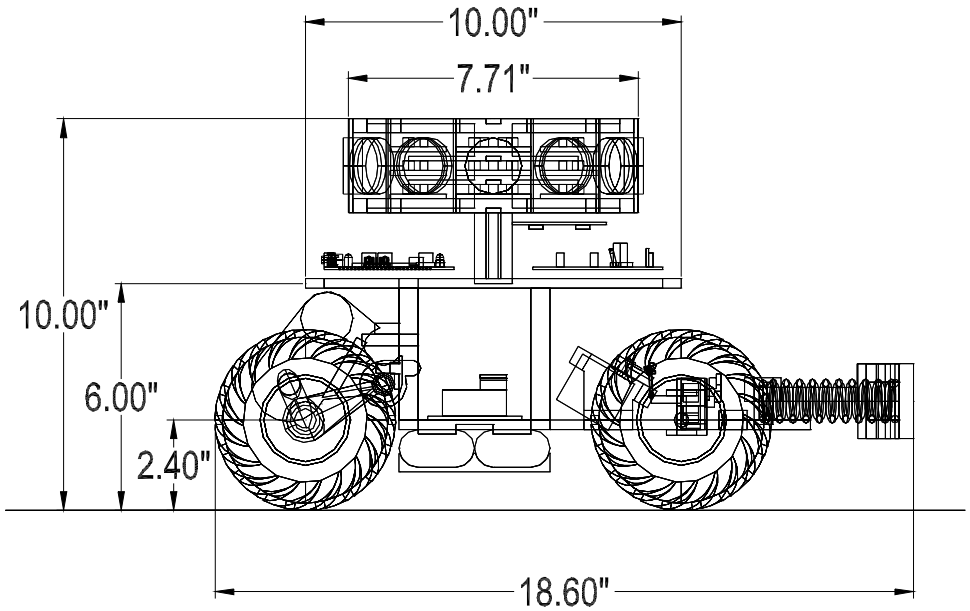

Fig. A.1 ROACH – Side View

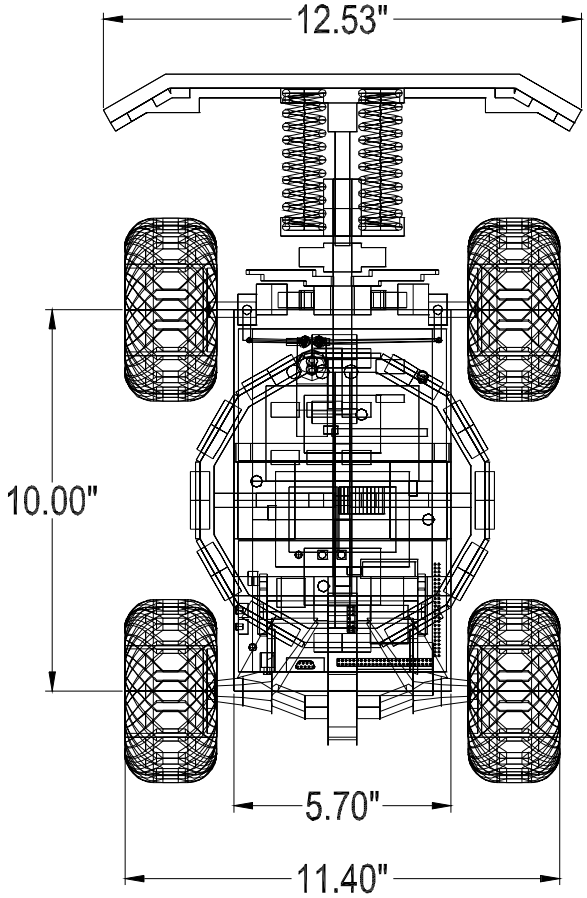

Fig. A.2 ROACH – Top View

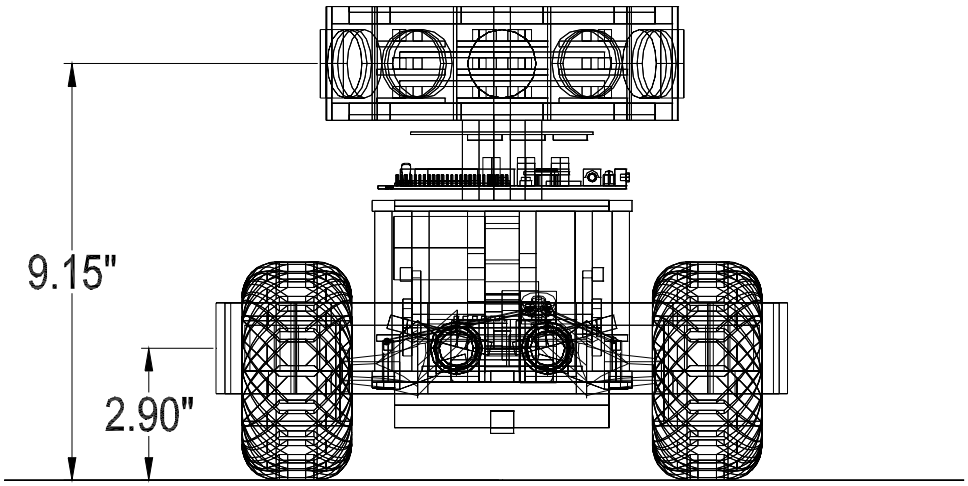

Fig. A.3 ROACH – Front View

APPENDIX B

LIST OF COMPONENTS

Component	Manufacturer	Part Name / Number	Approximate Price
1:10 Scale RC Car	Tamiya	Blackfoot Ford F-150 Ranger	$300
Receiver	Futaba	FP-R112JE	
Servo	Futaba	FP-S148	
Speed Controller	Dynamite	Power Pulse Speed Control	$80
Motor	Johnson		
Transducer	Polaroid	600 Series Instrument Grade Electrostatic Transducer	$30
Ultrasonic Circuit Board	Polaroid	6500 Series Sonar Ranging Module	$25
DSP Board	Analog Devices	ADSP-2181 EZ-KIT Lite	$75
Battery	Dynamite	DYNA-SPORT 1500	$20

Table B.1 ROCH Components

APPENDIX C

CONTROL ALGORITHM

```
/************************************************************************/
/*                  ASSEMBLER CODE - MOBILE ROBOT (ROACH)               */
/************************************************************************/

.module/ram/abs=0    sweet_robot;

/************************************************************************/
/*                              INITIAL SETUP                           */
/************************************************************************/

#include       <DSP.h>

#define        TSTEP              40     /* 40*30ns = 1200ns               */
#define        TLENGTH            10     /* 10*1200ns = 0.012ms            */
#define        READ_TIME          292    /* 292*0.012ms = 3.5ms            */
#define        TIME_CYCLE         1667   /* 1667*0.012ms = 20ms            */
#define        CYCLE_MAX          2500   /* 2500*0.012ms = 30ms            */
#define        SEVEN_CYCLES       11667  /* 11667_0.012ms = 140ms          */
#define        MIN_DISTANCE       75     /* 0.5ft                          */
#define        MAX_DISTANCE       1050   /* 1050*0.012ms = 12.6ms (7ft max) */
#define        ZERO               0
#define        ONE                1
#define        TWO                2
#define        THREE              3
#define        FOUR               4
#define        FIVE               5
#define        SIX                6
#define        SEVEN              7
#define        EIGHT              8
#define        NINE               9
#define        TEN                10
#define        ELEVEN             11
#define        TWELVE             12
#define        SERVO_PLUS         118

.var/dm/ram    clock;
.var/dm/ram    SV_clock;
.var/dm/ram    CT_clock;
.var/dm/ram    sonar_clock;
.var/dm/ram    CH1_time;
.var/dm/ram    CH2_time;
.var/dm/ram    CH1_flag;
.var/dm/ram    CH2_flag;
.var/dm/ram    sonar_flag;
.var/dm/ram    sonar_switch;
.var/dm/ram    sonar_set_new;
.var/dm/ram    sonar_set_old;
.var/dm/ram    neural_bypass;
.var/dm/ram    SERVO_TIME_;
.var/dm/ram    SERVO_PULSE_;
.var/dm/ram    CTRLR_TIME_;
.var/dm/ram    CTRLR_PULSE_;
.var/dm/ram    ECHO1;
.var/dm/ram    ECHO2;
.var/dm/ram    SONAR_NEW_;
.var/dm/ram    SONAR_OLD_;
.var/dm/ram    sonar_angle1;
.var/dm/ram    sonar_angle2;
.var/dm/ram    THETA_NEW_;
.var/dm/ram    THETA_OLD_;
.var/dm/ram    DEL_T_;

.global        SERVO_TIME_;
.global        SERVO_PULSE_;
.global        CTRLR_TIME_;
.global        CTRLR_PULSE_;
.global        SONAR_NEW_;
```

```
.global        SONAR_OLD_;
.global        THETA_NEW_;
.global        THETA_OLD_;
.global        DEL_T_;

.init          clock:          0;
.init          SERVO_TIME_:    0;
.init          SERVO_PULSE_:   0;
.init          CTRLR_TIME_:    0;
.init          CTRLR_PULSE_:   0;
.init          CH1_time:       0;
.init          CH2_time:       0;
.init          SV_clock:       0;
.init          CT_clock:       0;
.init          CH1_flag:       ZERO;
.init          CH2_flag:       ZERO;
.init          sonar_flag:     ZERO;
.init          sonar_switch:   ZERO;
.init          sonar_set_new:  ZERO;
.init          sonar_set_old:  ZERO;
.init          neural_bypass:  ONE;
.init          sonar_clock:    0;
.init          ECHO1:          0;
.init          ECHO2:          0;
.init          SONAR_NEW_:     MAX_DISTANCE;
.init          SONAR_OLD_:     MAX_DISTANCE;
.init          sonar_angle1:   TWELVE;
.init          sonar_angle2:   TWELVE;
.init          THETA_NEW_:     TWELVE;
.init          THETA_OLD_:     TWELVE;
.init          DEL_T_:         0;

/*****************************************************************************/
/*                                  NOTES                                    */
/*****************************************************************************/

/* Servo Maximum:                1.416ms (118)                               */
/* Servo 0-Position:             1.14ms  (95)                                */
/* Servo Minimum:                0.864ms (72)                                */
/* Controler FWD Maximum:        1.14ms  (95)                                */
/* Controler 0-position:         1.416ms (118)                               */
/* Controler BWD Maximum:        1.692ms (141)                               */
/* Distance Conversion:          Y(in) = 0.093 * X(count) - 1.5              */
/* Distance Conversion:          X(count) = (Y(in) + 1.5) / 0.093            */

/* PF0 (pin 25): BINH                                                        */
/* PF1 (pin 26): from CH1                                                    */
/* PF2 (pin 27): from CH2                                                    */
/* PF3 (pin 28): ECHO1                                                       */
/* PF4 (pin 29): ECHO2                                                       */
/* PF5 (pin 30): Mult1                                                       */
/* PF6 (pin 31): Mult2                                                       */
/* PF7 (pin 32): Mult3                                                       */
/* FL0 (pin 33): to Servo                                                    */
/* FL1 (pin 34): Alarm                                                       */
/* FL2 (pin 35): to Controller                                               */

/* Demulitplexer 000: 0 (pin 15)                                             */
/* Demulitplexer 100: 1 (pin 14) -> SONAR SET1                               */
/* Demulitplexer 010: 2 (pin 13) -> SONAR SET2                               */
/* Demulitplexer 110: 3 (pin 12) -> SONAR SET3                               */
/* Demulitplexer 001: 4 (pin 11) -> SONAR SET4                               */
/* Demulitplexer 101: 5 (pin 10) -> SONAR SET5                               */
/* Demulitplexer 011: 6 (pin 9)  -> SONAR SET6                               */
/* Demulitplexer 111: 7 (pin 7)                                              */

/* SONAR SET1: SENSOR 0 & SENSOR 6                                           */
```

```
/* SONAR SET2: SENSOR 3 & SENSOR 9                                          */
/* SONAR SET3: SENSOR 1 & SENSOR 7                                          */
/* SONAR SET4: SENSOR 4 & SENSOR 10                                         */
/* SONAR SET5: SENSOR 2 & SENSOR 8                                          */
/* SONAR SET6: SENSOR 5 & SENSOR 11                                         */

/***************************************************************************/
/*                          INTERRUPT VECTORS TABLE                        */
/***************************************************************************/

       jump start; rti; rti; rti;               {00: reset }
       rti; rti; rti; rti;                      {04: IRQ2 }
       rti; rti; rti; rti;                      {08: IRQL1}
       rti; rti; rti; rti;                      {0c: IRQL0 }
       rti; rti; rti; rti;                      {10: SPORT0 tx }
       rti; rti; rti; rti;                      {14: SPORT0 rx }
       rti; rti; rti; rti;                      {18: IRQE }
       rti; rti; rti; rti;                      {1c: BDMA }
       rti; rti; rti; rti;                      {20: SPORT1tx or IRQ1}
       rti; rti; rti; rti;                      {24: SPORT1rx or IRQ0}
       jump update_signal; rti; rti; rti;       {28: timer }
       rti; rti; rti; rti;                      {2c: power down }

/***************************************************************************/
/*                              Timer-Setup                                */
/***************************************************************************/

start:
       RESET FL1;
       AX0 = TSTEP;
       dm(TSCALE) = AX0;              /* 16*30ns = 480ns */
       AX1 = TLENGTH;
       dm(TPERIOD) = AX1;             /* 10*480ns = 0.0048ms */
       imask = b#0000000001;          /* enable TIMER */
       AX0 = b#0111101111100001;      /* PF1,PF2,PF3,PF4 = input */
       dm(PFTYPE) = AX0;
       AX1 = b#0000000000000000;
       dm(PFDATA) = AX1;              /* output = 0 */
       ena TIMER;
       AR = ZERO;
       dm(sonar_flag) = AR;
wait:   NOP;
       jump wait;

/***************************************************************************/
/*                             Timer_Reset                                 */
/***************************************************************************/

time_reset:
       AX1 = TLENGTH;
       dm(TCOUNT) = AX1;
       AR = ZERO;
       dm(clock) = AR;
       dm(CH1_time) = AR;
       dm(CH2_time) = AR;
       dm(SV_clock) = AR;
       dm(CT_clock) = AR;
       dm(CH1_flag) = AR;
       dm(CH2_flag) = AR;
       dm(sonar_flag) = AR;
       dm(sonar_clock) = AR;
       dm(ECHO1) = AR;
       dm(ECHO2) = AR;
       AR = TWELVE;
       dm(sonar_angle1) = AR;
       dm(sonar_angle2) = AR;
       RESET FL0;
```

```
        RESET FL2;
        AX1 = dm(PFDATA);
        AR = CLRBIT 0 OF AX1;           /* BINH */
        AR = CLRBIT 5 OF AR;            /* Demultiplexer 1 */
        AR = CLRBIT 6 OF AR;            /* Demultiplexer 2 */
        AR = CLRBIT 7 OF AR;            /* Demultiplexer 3 */
        dm(PFDATA) = AR;
        jump update_signal;

/*************************************************************************/
/*                              Pulse-In                                 */
/*************************************************************************/

update_signal:
        AX0 = dm(clock);
        AY0 = READ_TIME;
        AR = AX0 - AY0;
        IF GE jump echo_check;
        AX1 = dm(CH1_flag);
        AY1 = ZERO;
        AR = AX1 - AY1;
        IF NE jump skip_read_CH1;
        AX1 = dm(PFDATA);
        AR = TSTBIT 1 OF AX1;
        IF NE jump skip_read_CH1;
        AY0 = dm(CH1_time);
        AR = AX0 - AY0;
        dm(SERVO_TIME_) = AR;
        AR = ONE;
        dm(CH1_flag) = AR;
skip_read_CH1:
        AX1 = dm(CH2_flag);
        AY1 = ZERO;
        AR = AX1 - AY1;
        IF NE jump skip_read_CH2;
        AX1 = dm(CH2_time);
        AY1 = ZERO;
        AR = AX1 - AY1;
        IF NE jump skip_wait_CH2;
        AX1 = dm(PFDATA);
        AR = TSTBIT 2 OF AX1;
        IF EQ jump skip_read_CH2;
        dm(CH2_time) = AX0;
skip_wait_CH2:
        AX1 = dm(PFDATA);
        AR = TSTBIT 2 OF AX1;
        IF NE jump skip_read_CH2;
        AY0 = dm(CH2_time);
        AR = AX0 - AY0;
        dm(CTRLR_TIME_) = AR;
        AR = ONE;
        dm(CH2_flag) = AR;
skip_read_CH2:
        jump add_clock;

/*************************************************************************/
/*             Selection Block  [ Sonar-INIT - Sonar-ECHO ]              */
/*************************************************************************/

echo_check:
        AX1 = dm(sonar_flag);
        AY1 = ZERO;
        AR = AX1 - AY1;
        IF EQ jump send_sonar_pulse;
        AR = dm(sonar_clock);
        dm(ECHO1) = AR;
        dm(ECHO2) = AR;
```

```
        AR = AR + 1;
        dm(sonar_clock) = AR;
        AX1 = dm(ECHO1);
        AY1 = MIN_DISTANCE;
        AR = AX1 - AY1;
        IF LT jump send_sonar_pulse;
        AR = AX1 - AY1;
        IF NE jump check_ECHO1;
        AX1 = dm(PFDATA);
        AR = SETBIT 0 OF AX1;
        dm(PFDATA) = AR;
check_ECHO1:
        AX1 = dm(PFDATA);
        AR = TSTBIT 3 OF AX1;
        IF EQ jump check_ECHO2;
        AX1 = dm(ECHO1);
        AY1 = dm(SONAR_NEW_);
        AR = AX1 - AY1;
        IF GT jump sonar_flag_down;
        dm(SONAR_NEW_) = AX1;
        AR = dm(sonar_angle1);
        dm(THETA_NEW_) = AR;
        AR = dm(sonar_switch);
        dm(sonar_set_new) = AR;
        jump sonar_flag_down;
check_ECHO2:
        AX1 = dm(PFDATA);
        AR = TSTBIT 4 OF AX1;
        IF EQ jump send_sonar_pulse;
        AX1 = dm(ECHO2);
        AY1 = dm(SONAR_NEW_);
        AR = AX1 - AY1;
        IF GT jump sonar_flag_down;
        dm(SONAR_NEW_) = AX1;
        AR = dm(sonar_angle2);
        dm(THETA_NEW_) = AR;
        AR = dm(sonar_switch);
        dm(sonar_set_new) = AR;
        jump sonar_flag_down;
sonar_flag_down:
        AR = ZERO;
        dm(sonar_flag) = AR;
        jump send_sonar_pulse;

/*************************************************************************/

send_sonar_pulse:
        AX0 = dm(clock);
        AY0 = MAX_DISTANCE;
        AR = AX0 - AY0;
        IF GT jump skip_send_pulse;
        AY0 = READ_TIME;
        AR = AX0 - AY0;
        IF NE jump send_driving_pulse;
        AR = ZERO;
        dm(SV_clock) = AR;
        dm(CT_clock) = AR;
sonar_set1:

        jump sonar_set2;

        AX1 = dm(sonar_switch);
        AY1 = ZERO;
        AR = AX1 - AY1;
        IF NE jump sonar_set2;
        AX1 = dm(PFDATA);
        AR = SETBIT 5 OF AX1;
```

```
        AR = CLRBIT 6 OF AR;
        AR = CLRBIT 7 OF AR;
        dm(PFDATA) = AR;
        AR = ZERO;
        dm(sonar_angle1) = AR;
        AR = SIX;
        dm(sonar_angle2) = AR;
        AR = ONE;
        dm(sonar_flag) = AR;
        AR = ZERO;
        dm(sonar_clock) = AR;
        jump switch;
sonar_set2:
        AX1 = dm(sonar_switch);
        AY1 = ONE;
        AR = AX1 - AY1;
        IF NE jump sonar_set3;
        AX1 = dm(PFDATA);
        AR = CLRBIT 5 OF AX1;
        AR = SETBIT 6 OF AR;
        AR = CLRBIT 7 OF AR;
        dm(PFDATA) = AR;
        AR = THREE;
        dm(sonar_angle1) = AR;
        AR = NINE;
        dm(sonar_angle2) = AR;
        AR = ONE;
        dm(sonar_flag) = AR;
        AR = ZERO;
        dm(sonar_clock) = AR;
        jump switch;
sonar_set3:
        AX1 = dm(sonar_switch);
        AY1 = TWO;
        AR = AX1 - AY1;
        IF NE jump sonar_set4;
        AX1 = dm(PFDATA);
        AR = SETBIT 5 OF AX1;
        AR = SETBIT 6 OF AR;
        AR = CLRBIT 7 OF AR;
        dm(PFDATA) = AR;
        AR = ONE;
        dm(sonar_angle1) = AR;
        AR = SEVEN;
        dm(sonar_angle2) = AR;
        AR = ONE;
        dm(sonar_flag) = AR;
        AR = ZERO;
        dm(sonar_clock) = AR;
        jump switch;
sonar_set4:
        AX1 = dm(sonar_switch);
        AY1 = THREE;
        AR = AX1 - AY1;
        IF NE jump sonar_set5;
        AX1 = dm(PFDATA);
        AR = CLRBIT 5 OF AX1;
        AR = CLRBIT 6 OF AR;
        AR = SETBIT 7 OF AR;
        dm(PFDATA) = AR;
        AR = FOUR;
        dm(sonar_angle1) = AR;
        AR = TEN;
        dm(sonar_angle2) = AR;
        AR = ONE;
        dm(sonar_flag) = AR;
        AR = ZERO;
```

```
        dm(sonar_clock) = AR;
        jump switch;
sonar_set5:
        AX1 = dm(sonar_switch);
        AY1 = FOUR;
        AR = AX1 - AY1;
        IF NE jump sonar_set6;
        AX1 = dm(PFDATA);
        AR = SETBIT 5 OF AX1;
        AR = CLRBIT 6 OF AR;
        AR = SETBIT 7 OF AR;
        dm(PFDATA) = AR;
        AR = TWO;
        dm(sonar_angle1) = AR;
        AR = EIGHT;
        dm(sonar_angle2) = AR;
        AR = ONE;
        dm(sonar_flag) = AR;
        AR = ZERO;
        dm(sonar_clock) = AR;
        jump switch;
sonar_set6:
        AX1 = dm(sonar_switch);
        AY1 = FIVE;
        AR = AX1 - AY1;
        IF NE jump bypass_check;
        AX1 = dm(PFDATA);
        AR = CLRBIT 5 OF AX1;
        AR = SETBIT 6 OF AR;
        AR = SETBIT 7 OF AR;
        dm(PFDATA) = AR;
        AR = FIVE;
        dm(sonar_angle1) = AR;
        AR = ELEVEN;
        dm(sonar_angle2) = AR;
        AR = ONE;
        dm(sonar_flag) = AR;
        AR = ZERO;
        dm(sonar_clock) = AR;
        jump switch;

/*************************************************************************/
/*              Selection Block [ Crash Avoidance Algorithm ]            */
/*************************************************************************/

bypass_check:
        AX1 = dm(sonar_switch);
        AY1 = SIX;
        AR = AX1 - AY1;
        IF NE jump switch;
        AX1 = TWELVE;
        AY1 = dm(THETA_NEW_);
        AR = AX1 - AY1;
        IF EQ jump neural_bypass_on;
        AX1 = dm(neural_bypass);
        AY1 = ONE;
        AR = AX1 - AY1;
        IF EQ jump neural_bypass_off;
calculation_DEL_T:
        AX1 = dm(sonar_set_new);
        AY1 = dm(sonar_set_old);
        MY1 = TIME_CYCLE;
        AR = AX1 - AY1;
        SR0 = AR;
        MR  = SR0 * MY1 (ss);
        AY0 = SEVEN_CYCLES;
        AR = MR0 + AY0;
```

```
        AX1 = AR;
        AR = dm(SONAR_NEW_);
        AY1 = dm(SONAR_OLD_);
        AR = AR - AY1;
        SR = LSHIFT AR BY -15 (HI);
        AY1 = SR1;
        AR = MR0 + AY1;
        SR = ASHIFT AR BY -1 (HI);
        AY0 = SR1;
        AR = AX1 + AY0;
        dm(DEL_T_) = AR;
        jump call_neural_net;
call_neural_net:
        call main_;
        jump store_old_data;
neural_bypass_on:
        AR = ONE;
        dm(neural_bypass) = AR;
        jump bypass_signal;
neural_bypass_off:
        AR = ZERO;
        dm(neural_bypass) = AR;
        jump bypass_signal;
bypass_signal:
        AR = dm(SERVO_TIME_);
        dm(SERVO_PULSE_) = AR;
        AR = dm(CTRLR_TIME_);
        dm(CTRLR_PULSE_) = AR;
store_old_data:
        AR = dm(SONAR_NEW_);
        dm(SONAR_OLD_) = AR;
        AR = MAX_DISTANCE;
        dm(SONAR_NEW_) = AR;
        AR = dm(THETA_NEW_);
        dm(THETA_OLD_) = AR;
        AR = TWELVE;
        dm(THETA_NEW_) = AR;
        AR = dm(sonar_set_new);
        dm(sonar_set_old) = AR;
        jump switch;
switch:
        AR = dm(sonar_switch);
        AR = AR + 1;
        dm(sonar_switch) = AR;
        AX1 = dm(sonar_switch);
        AY1 = SIX;
        AR = AX1 - AY1;
        IF LE jump send_driving_pulse;
        AR = ZERO;
        dm(sonar_switch) = AR;
        jump send_driving_pulse;

/*************************************************************************/
/*                              Pulse-Out                                */
/*************************************************************************/

send_driving_pulse:
        AX1 = dm(SV_clock);
        AY1 = dm(SERVO_PULSE_);
        AR = AX1 - AY1;
        IF GE RESET FL0;
        IF LT SET FL0;
        AX1 = dm(CT_clock);
        AY1 = dm(CTRLR_PULSE_);
        AR = AX1 - AY1;
        IF GE RESET FL2;
        IF LT SET FL2;
```

```
        AR = dm(SV_clock);
        AR = AR + 1;
        dm(SV_clock) = AR;
        AR = dm(CT_clock);
        AR = AR + 1;
        dm(CT_clock) = AR;
skip_send_pulse:
        AX0 = dm(clock);
        AY0 = MAX_DISTANCE;
        AR = AX0 - AY0;
        IF LE jump add_clock;
        AX1 = dm(PFDATA);
        AR = TSTBIT 1 OF AX1;
        IF NE jump time_reset;

/*************************************************************************/
/*                                Stand-By                               */
/*************************************************************************/

add_clock:
        AX0 = dm(clock);
        AY0 = CYCLE_MAX;
        AR = AX0 - AY0;
        IF GT rti;
        AR = AX0 + 1;
        dm(clock) = AR;
        rti;

/*************************************************************************/
/*                            CRASH AVOIDANCE                            */
/*************************************************************************/

main_:

/*               Contents of the crash avoidance algorithm.              */

        rts;

/*************************************************************************/
/*                                  END                                  */
/*************************************************************************/

.endmod;
```

APPENDIX D

SIMPLE CRASH AVOIDANCE ALGORITHM

(In C Language: as an External Program)

```
/*************************************************************************/
/*                  C CODE – A DRUNKEN SAILOR WALK (ROACH)                */
/*************************************************************************/

#define        TEST_DISTANCE1        338
#define        TEST_DISTANCE2        338
#define        SERVO_PLUS            118
#define        SERVO_MINUS           72
#define        CTRLR_FWD_MAX         95
#define        CTRLR_ZERO            118
#define        CTRLR_BWD_MAX         141


extern int SERVO_TIME;
extern int SERVO_PULSE;
extern int CTRLR_TIME;
extern int CTRLR_PULSE;
extern int SONAR_NEW;
extern int SONAR_OLD;

INTELLIGENCE ()
{
       if (SONAR_NEW < TEST_DISTANCE1) AVOIDWALL();
       else
       if (SONAR_NEW > TEST_DISTANCE2) FINDWALL();
       else
                                       ZERO_POSITION();
       return;
}

ZERO_POSITION ()
{
       SERVO_PULSE = SERVO_TIME;
       CTRLR_PULSE = CTRLR_TIME;
       asm("RESET FL1;");
       return;
}

FINDWALL ()
{
       SERVO_PULSE = SERVO_PLUS;
       CTRLR_PULSE = CTRLR_TIME;
       asm("SET FL1;");
       return;
}

AVOIDWALL ()
{
       SERVO_PULSE = SERVO_MINUS;
       CTRLR_PULSE = CTRLR_TIME;
       asm("TOGGLE FL1;");
       return;
}
```

APPENDIX E

SIMPLE CRASH AVOIDANCE ALGORITHM

(In Assembly Language)

```
/*****************************************************************************/
/*               ASSEMBLER CODE – A DRUNKEN SAILOR WALK (ROACH)              */
/*****************************************************************************/

main_:
        ay1=dm(SONAR_NEW_);
        ax1=337;
        af=ay1-ax1;
        if gt jump mainL2_;
        call AVOIDWALL_;
        jump mainL1_;
mainL2_:
        ay1=dm(SONAR_NEW_);
        ax1=338;
        af=ay1-ax1;
        if le jump mainL4_;
        call FINDWALL_;
        jump mainL1_;
mainL4_:
        call ZERO_POSITION_;
mainL1_:
        rts;

ZERO_POSITION_:
        ax1=dm(SERVO_TIME_);
        dm(SERVO_PULSE_)=ax1;
        ax1=dm(CTRLR_TIME_);
        dm(CTRLR_PULSE_)=ax1;
        RESET FL1;
        rts;

FINDWALL_:
        ax1=118;
        dm(SERVO_PULSE_)=ax1;
        ax1=dm(CTRLR_TIME_);
        dm(CTRLR_PULSE_)=ax1;
        SET FL1;
        rts;

AVOIDWALL_:
        ax1=72;
        dm(SERVO_PULSE_)=ax1;
        ax1=dm(CTRLR_TIME_);
        dm(CTRLR_PULSE_)=ax1;
        TOGGLE FL1;
        rts;
```

APPENDIX F

SIMPLE CRASH AVOIDANCE ALGORITHM

(In Assembly Language: as an External Program)

```
/*************************************************************************/
/*               ASSEMBLER CODE – A DRUNKEN SAILOR WALK (ROACH)            */
/*************************************************************************/

!              Analog Devices ADSP21XX
.MODULE/RAM    _track_;
!gcc_compiled
.external      SONAR_NEW_;
.external      AVOIDWALL_;
.external      FINDWALL_;
.external      ZERO_POSITION_;


.entry  main_;
main_:
!       FUNCTION PROLOGUE: main
        mr1=toppcstack;        ! get return address
        si=m4;
        m4=i4;         ! new frame ptr <= old stack ptr
        m5=-1;
        dm(i4,m5)=si;  ! save old frame pointer
        dm(i4,m5)=mr1; ! save return address
!               saving registers:
        dm(i4,m5)=ax1;
!       END FUNCTION PROLOGUE: main
        ay1=dm(SONAR_NEW_);
        ax1=337;
        af=ay1-ax1;
        if gt jump mainL2_;
        call AVOIDWALL_;
        jump mainL1_;
mainL2_:
        ay1=dm(SONAR_NEW_);
        ax1=338;
        af=ay1-ax1;
        if le jump mainL4_;
        call FINDWALL_;
        jump mainL1_;
mainL4_:
        call ZERO_POSITION_;
mainL1_:
!       FUNCTION EPILOGUE: main
        i6=m4;
        m5=-1;
        si=dm(i6,m5);  ! old frame pointer
        mr1=dm(i6,m5); ! return address
!               restoring registers:
        ax1=dm(i6,m5);
        i4=m4;  ! reset stack pointer
        i6=mr1;
        m4=si;  ! reset frame pointer
!       END FUNCTION EPILOGUE: main
        jump (i6);

.external      SERVO_PULSE_;
.external      SERVO_TIME_;
.external      CTRLR_PULSE_;
.external      CTRLR_TIME_;
.entry  ZERO_POSITION_;
ZERO_POSITION_:
!       FUNCTION PROLOGUE: ZERO_POSITION
        mr1=toppcstack;        ! get return address
        si=m4;
        m4=i4;         ! new frame ptr <= old stack ptr
        m5=-1;
        dm(i4,m5)=si;  ! save old frame pointer
        dm(i4,m5)=mr1; ! save return address
```

```
!               saving registers:
        dm(i4,m5)=ax1;
!       END FUNCTION PROLOGUE: ZERO_POSITION
        ax1=dm(SERVO_TIME_);
        dm(SERVO_PULSE_)=ax1;
        ax1=dm(CTRLR_TIME_);
        dm(CTRLR_PULSE_)=ax1;
!APP
        RESET FL1;
!NO_APP
!       FUNCTION EPILOGUE: ZERO_POSITION
        i6=m4;
        m5=-1;
        si=dm(i6,m5);  ! old frame pointer
        mr1=dm(i6,m5); ! return address
!               restoring registers:
        ax1=dm(i6,m5);
        i4=m4;  ! reset stack pointer
        i6=mr1;
        m4=si;  ! reset frame pointer
!       END FUNCTION EPILOGUE: ZERO_POSITION
        jump (i6);

.entry  FINDWALL_;
FINDWALL_:
!       FUNCTION PROLOGUE: FINDWALL
        mr1=toppcstack;        ! get return address
        si=m4;
        m4=i4;          ! new frame ptr <= old stack ptr
        m5=-1;
        dm(i4,m5)=si;  ! save old frame pointer
        dm(i4,m5)=mr1; ! save return address
!               saving registers:
        dm(i4,m5)=ax1;
!       END FUNCTION PROLOGUE: FINDWALL
        ax1=118;
        dm(SERVO_PULSE_)=ax1;
        ax1=dm(CTRLR_TIME_);
        dm(CTRLR_PULSE_)=ax1;
!APP
        SET FL1;
!NO_APP
!       FUNCTION EPILOGUE: FINDWALL
        i6=m4;
        m5=-1;
        si=dm(i6,m5);  ! old frame pointer
        mr1=dm(i6,m5); ! return address
!               restoring registers:
        ax1=dm(i6,m5);
        i4=m4;  ! reset stack pointer
        i6=mr1;
        m4=si;  ! reset frame pointer
!       END FUNCTION EPILOGUE: FINDWALL
        jump (i6);

.entry  AVOIDWALL_;
AVOIDWALL_:
!       FUNCTION PROLOGUE: AVOIDWALL
        mr1=toppcstack;        ! get return address
        si=m4;
        m4=i4;          ! new frame ptr <= old stack ptr
        m5=-1;
        dm(i4,m5)=si;  ! save old frame pointer
        dm(i4,m5)=mr1; ! save return address
!               saving registers:
        dm(i4,m5)=ax1;
!       END FUNCTION PROLOGUE: AVOIDWALL
```

```
        ax1=72;
        dm(SERVO_PULSE_)=ax1;
        ax1=dm(CTRLR_TIME_);
        dm(CTRLR_PULSE_)=ax1;
!APP
        TOGGLE FL1;
!NO_APP
!       FUNCTION EPILOGUE: AVOIDWALL
        i6=m4;
        m5=-1;
        si=dm(i6,m5);  ! old frame pointer
        mr1=dm(i6,m5); ! return address
!               restoring registers:
        ax1=dm(i6,m5);
        i4=m4;  ! reset stack pointer
        i6=mr1;
        m4=si;  ! reset frame pointer
!       END FUNCTION EPILOGUE: AVOIDWALL
        jump (i6);

.ENDMOD;
```

APPENDIX G

CONTROL ALGORITHM - C INTERFACE

```
/*****************************************************************************/
/*                                C-Interface                                */
/*****************************************************************************/

virtual_insanity:
        mr1 = toppcstack;
        mr0 = MAGIC_NUMBER;
        call ___lib_save_large_frame;
        call main_;
        mr0 = MAGIC_NUMBER;
        call ___lib_restore_large_frame;
        rts;
```