DESIGN AND INTEGRATION OF HARDWARE AND SOFTWARE TO IMPLEMENT AN AUTONOMOUS VEHICLE

By

ENRIQUE MARTIN ACUÑA

Bachelor in Electrical Engineering
University of Costa Rica
San José, Costa Rica
1991

Licenciado in Electrical Engineering
University of Costa Rica
San José, Costa Rica
1993

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1996

DESIGN AND INTEGRATION OF HARDWARE AND

SOFTWARE TO IMPLEMENT AN

AUTONOMOUS VEHICLE

Thesis Approved:

Thesis Adviser

Dean of the Graduate College

ACKNOWLEDGMENTS

Countless thanks, infinitas gracias, muilto obrigado to my adviser and professor Dr. Eduardo Misawa for trusting me and giving me the great honor of working in this research, which I consider a very important and necessay contribution to our society, specially after having been physically disabled; thanks to him for granting me all his invaluable time not only while close at OSU, but also while far away during those months I was in my country in rehabilitation.

Thanks to Dr. Martin Hagan and Dr. Rajni Patel for serving on my committee.

Thanks to Dr. Marvin Stone whose help was a key to the development of this work.

Thanks to the secretaries of Electrical and Computer Engineering and Mechanical and Aerospace Engineering Schools and MAERL (West Lab) for their permanent cooperation.

Thanks to Eng. Jorge M. Páez, Head of the Electrical Engineering School of the University of Costa Rica, for his support to my work while in my country.

Thanks to the engineer John Newton to whom I admire and consider and excellent example of an engineer by heart and vocation.

Thanks to the engineers Mike Veldman and Gordon Cougar who were always there to offer their wise advice and support.

Thanks to my sponsor, the US Agency for the International Development, and the Office of International Programs of OSU, specially to Christie Millis, my international adviser, for building and believing in a better world and for making of my scholarship's program of study a most important experience in my life.

Thanks to my parents, Carmen Acosta and Enrique Acuña, peace and treasure of my life, to whom I dedicate this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Background

"The concepts of intelligent and autonomous control have sparked much discussion and debate in the last few years" [27]. The autonomous "control of mobile robots has recently been the subject of intense research. A variety of approaches from engineering and artificial intelligence have been used to study different aspects of robot control and navigation. Many of these efforts are aimed at the development of simple intelligent behavior, especially when the robot is operating in an unknown environment" [9].

The Problem in Study

A relative success of classical control theory has led many of the above research activities "to focus on higher-level tasks, while ignoring the details of low-level control. However, the process of actually implementing a model in hardware has been found challenging by those who have tried. Frequently the main difficulties arise at rather low-levels of control" [9].

The present research project deals with the problem on how to implement a mobile robot, namely an autonomous vehicle, taking into consideration all levels of control, those in hardware and software.

Objectives of the Study

The general objective of the present research is the design and integration of hardware and software to implement an autonomous vehicle, from now on known as AV.

The AV is thought to be implemented as a distributed system that comprises five main modules: the Vision Module, the Propulsion Module, the Supervisor Module, the communications networks and the gateway to communicate two different networks.

The hardware of this AV will consist of four main parts:

1) Physical platform (which is a motorized wheelchair that includes the wheelchairs own electronics and its mechanical part; it is shown in figure 1).

Figure 1. Wheelchair Chassis [19]

2) Additional electronics needed for desired control (this has been installed in a previous project [19] and is modified in the present work).

3) Three microcontrollers (one for the vision module, one for the propulsion module and one for the gateway) and a Personal Computer (PC) to install the supervisor module.

4) Controller Area Network (or CAN, which is introduced in Chapter II) and Serial Communications Section (SCS).

Each one of the four computers has its own specific software, and each one of these programs includes the corresponding protocol required for the network it is serving.

The specific objectives of the research are the following:

1) To replace the two microcontrollers (mounted in the wheelchair in a previous project [19]) and do all the necessary modifications in hardware and software dictated by the new controllers. The two microcontrollers are first prototypes of the AV's Propulsion and Vision modules and were tested independently; they are similar microcontrollers to the new ones, but unable to handle network communications.

2) To develop the software required to install the CAN.

3) To develop the software for the serial communications between the PC and the gateway.

4) To develop the software needed by the gateway in order to communicate both networks (PC-gateway network and CAN).

5) To develop the Supervisor Module that can be used to test the performance of the AV after the integration stage of all its components. This Supervisor is designed using Artificial Intelligence techniques specifically Fuzzy Logic. To achieve this objective the AV has to perform one or several given tasks. It is chosen that the supervisor implement two tasks the avoidance of collisions with obstacles and the following of a moving object.

Significance of the Study

The importance of designing and implementing the AV is explained by the following potential applications:

1) One of the main characteristics of the AV is its "ability to grow", in other words, since it is a distributed system that implements the CAN, more microcontrollers can be added to make of the AV a robot that can perform more tasks and even more complex, such as speech processing and the movement of a "human-like" arm, that also can be mounted in the AV.

2) Other main characteristic of the AV is its modular structure that allows the engineer to reconfigure the AV. For instance the five modules may all be moved to other different physical platform specifically designed so the AV can be used for sea or space exploration. Of course the reconfiguration may require some minor modifications.

3) Manufacturing plants and industries that handle toxic waste disposals are using mobile robots nowadays. These robots are flexible to some extent, however it is necessary that they become able to deal with unknown and dynamic environments and make their own decisions without human guidance [2].

4) The AV can be used for protecting human life in toxic environments, where the supervision of human beings is highly dangerous and eliminating magnetic stripes or grooves used to guide mobile robots in manufacturing environments, since the AV may be able to keep an updated map of the environment.

5) In hospitals the AV can be used by handicapped people (blind, spinal cord injured, etc.).

6) The AV may be used as a tool to keep developing theoretical foundations in the analysis, design and use of distributed computer systems in automatic and intelligent control.

Scope and Limitations

The AV is potentially able of performing several different tasks as it is emphasized in the last section "Significance of the Study". However, to evaluate the performance of the AV, as a whole, after the integration of all its components, two tasks are chosen: the following of a moving object and the avoidance of obstacle collisions. Therefore this research project comes to an end with the development of the Supervisor Module, that demonstrates that the AV is able to perform the given two tasks.

As a parallel experiment, the possibility of developing other supervisor that implements other tasks is also explored in this research. As it will be explained in Chapter II of Literature Review, Andujar [2] started to conceive the general structure of the AV and produced a simulation software for the AV, that is later improved by Shanley [24]. So the software designed by Andujar and Shanley is modified in the present research in order to use their supervisor module and adjust it to the AV all of this, again, with the objective of testing its performance and also of taking advantage of their work.

Hypothesis

The hypothesis to be addressed by this research work are explained next.

It is possible to implement an autonomous vehicle (AV) as a distributed system in which the Vision and Propulsion modules are controlled or "supervised" by a Supervisor module through the communications networks and the gateway. And also it is possible to design the Supervisor Module using Artificial Intelligence techniques, specifically Fuzzy Logic, to enable the AV to execute two main tasks: the following of a moving object and the avoidance of collisions with obstacles.

CHAPTER II

REVIEW OF THE LITERATURE

Previous Work

It is possible to say that the conception of the AV and the definition of other aspects for its development have been considered in four closely related research projects at Oklahoma State University:

1) "Autonomous Vehicle Control using Fuzzy Inference and a Fast Path Planning Algorithm" [2] in 1992 by Ricardo Andujar. As explained in his work, Andujar adopts the generic architecture for an autonomous mobile robot (proposed by other authors) and conceives the robot to consist of three modules, the Supervisor, the Vision and the Propulsion, that must conform to a CAN architecture (as shown in figure 2). Andujar develops a simulation software for this system and leaves the following contributions:

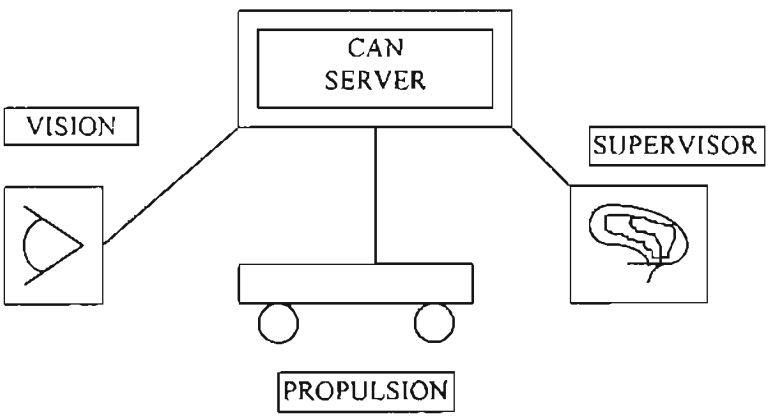


Figure 2. Generic Architecture for an Autonomous Vehicle [2].

- A fast sub-optimal path planner based on an environment map.

- Fast adaptive environment mapping.

- Complex planning capabilities while maintaining high reactivity using fuzzy logic in conjunction with the path planner.

2) "Environment Mapping and Adaptive Fuzzy Logic Control for an Autonomous Vehicle" [24] in 1994, by Robert Shanley. The main contribution of Shanley is to improve the software produced by Andujar by doing the following:

- The map used by Andujar, which was a pixel based screen map, is replaced by a Bit Field Map, since the robot will not have a computer screen.

- Two new fuzzy logic controllers are implemented for the supervisor, one is a modified version of Andujar's controller, and the other uses fuzzy logic adaptation routines.

3) "Ultrasonic Ranging System of an Obstacle-Avoidance Robot" [13] in 1993, Chun-Lin Lee; who studies and presents an implementation of the sensors that are going to be used by the AV in its vision module.

4) "Propulsion Design and Implementation for an Autonomous Vehicle" [19] in 1994, by John Newton, who prepares the physical platform of the AV by modifying a motorized wheelchair towards a vision guided AV. Newton also implements the first prototypes of the AV's Propulsion and Vision as independent modules and for this he uses two microcontrollers Motorola M68HC11EVB.

Distributed Computer System and Controller Area Network

The AV is thought as a distributed system. This term "has been used to define a wide range of computers -from systems connected by wide area networks, to those connected by local area networks, and even to distributed-memory parallel systems. What they have in common is that they contain multiple processing units, each of which can be engaged in its own activities while cooperating with other units in some computational task" [7]. This type of computer system architecture is becoming more attractive to researchers, because of the effectiveness, flexibility and cost that it offers [30].

One of the main factors that characterizes a distributed system is its interconnection structure which determines the network topology [7] and requires the definition of a communications architecture. "The Open System Interconnect (OSI) was developed by the International Organization for Standardization (ISO) in 1984 as a model of a computer communications architecture. The J1939 is high speed communications network structured into several parts based on the OSI model. The J1939 uses a protocol (set of rules to structure the information to be transferred) known as Controller Area Network or CAN ()14, presented in the ISO 11898 amended publication, which is a CAN 2.0 High Speed Proposal for an International Standard (that was in preparation as of December 1993) [22].

The AV is conceived to implement a CAN architecture, so the robot can use three independent computer boards communicating through a communications network tailored specifically for control communications. The Society of Automotive Engineers (SAE) is working on a standard framework for a CAN. The document used in this research corresponds to the Series of SAE Recommended Practices that have been developed by the Truck and Bus Control and Communications Network Subcommittee of the Truck and Bus Electrical and Electronics Committee. The objectives of the Subcommittee are to develop information reports, recommended practices, and standards concerned with the requirements design and usage of devices which transmit electronic signals and control information among vehicle components [21], [22], [23].

The Microcontrollers

The microcontrollers Motorola M68HC11EVB, already mounted in the wheelchair by Newton, are to be replaced by the ENAT boards, by Dearborn Group, Inc. To do this substitution it is necessary the study of both controllers.

The M68HC11 or simply HC11 is a family of 8-bit processors potentially able to address 64 Kb of memory in its expanded mode, when the bus becomes available outside the chip. However all 64 Kb is not available for actual memory use, because of memory mapped input and output. In the Evaluation Board (EVB) the user RAM has only 8 Kb available. The HC11 has six major ports and six addressing modes. It

uses a monitor program called BUFFALO that helps to load, test, and debug the user's program, but BUFFALO is not active at all when the program is running. In memory mapping used by the HC11 all instructions that can deal with memory also can deal with input and output. There are provisions for 18 different interrupts in the HC11 and 3 fatal interrupts: 15 work through the interrupt mask bit and 3 are special, bypassing the interrupt mask. Only one maskable interrupt can be processed at one time [14], [15], [16], [17].

The ENAT or Extended CAN Network Analysis Tool is a small double board controller intended to aid in the development and analysis of networks using the Intel 82527 CAN network interface. The base board is called the DNAT board and the daughter board is the EV82527 CAN adapter board.

The DNAT board has an M68HC11A0 microprocessor, 32 Kb of EPROM, 32 Kb of Non-volatile RAM, and an RS-232 communication port at 9600 baud to the host PC [3], [11]. The block, layout and electronic diagrams for the ENAT board are presented in appendix G .

Fuzzy Control

Fuzzy control implies the understanding of the theory of fuzzy relations and the calculus of fuzzy if-then rules or, simply, the calculus of fuzzy rules. In a typical way a fuzzy rule is expressed as "if X is A then Y is B" where X and Y are variables and A and B are their linguistic values.

Most of the experiences of the human being are stored in his memory in the form of fuzzy rules and almost all of human reasoning involves a manipulation of such rules. The role model for the calculus of fuzzy rules is the human mind [28]. By definition a robot is a "mechanical device operating in a seemingly human way" [29] and as such the AV has a very convenient and adequate tool in Fuzzy Logic to implement its "human-like mind", in other words the Supervisor Module.

The fuzzy logic controller (FLC) gives a control output whose production has been presented in the literature with different methods [4], [10], [12]. The method implemented in this research is known as the "Quality Method" [10] or Modified-Center-Average-Defuzzifier" (28).

Design Methodology for Autonomous Robots

When autonomous robots operate in rich, dynamic environments they must be able to utilize effectively and coordinate their limited physical and computational resources. As complexity increases, there appears a problem: interactions between behaviors also increase to the point where it becomes difficult to predict the system's overall behavior, besides the system understandability decreases and it is not possible to ensure that the robots will achieve their tasks. To solve this problem some methodologies to design robot systems have been developed like the Task Control Architecture (TCA), NASREM, the RAPs and others [25]. However this research project presents a methodology for the design and integration of the hardware and software of the AV that grows, in a rather heuristic way, with the process of conceiving and characterizing the AV itself, although always following the scientific method presented with terms and tools developed by the author in a previous thesis project [1].

CHAPTER III

METHODOLOGY FOR THE DEVELOPMENT OF THE AV

This chapter presents a systematic vision of the methodology used in this research and suggested by the author. The terms and tools presented here are also extended and applied, for the first time, to incorporate the research work of Andujar [2] , Shanley [24] and Newton [19], in order to harmonize this project with its historical antecedents and to present a global view of the development of the AV since its beginning.

The general scheme of the development of the AV can be represented as shown in figure 3.

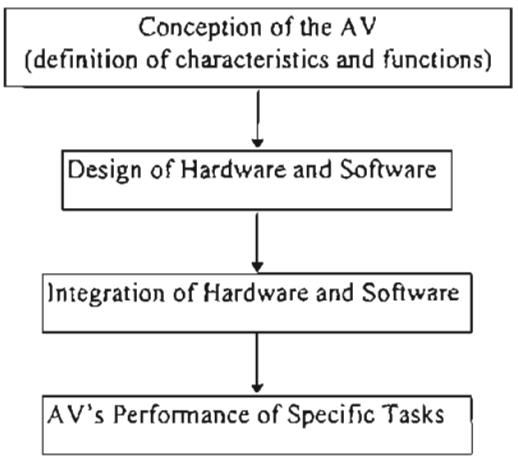


Figure 3. General Scheme for AV's Development

Now a more detailed scheme is presented in figure 4. Each box represents a process, that is to say the process of formulating a hypothesis for the design of the given subsystem, meaning by subsystem any system, namely a set of related things or parts of the AV with unity and its own functions and interactions clearly defined with other subsystems of its surroundings. There are two main types of

subsystems: the hardware and the software, therefore examples of subsystems may be: AV's simulation software, physical platform, Vision Module's software and Propulsion Module's hardware.

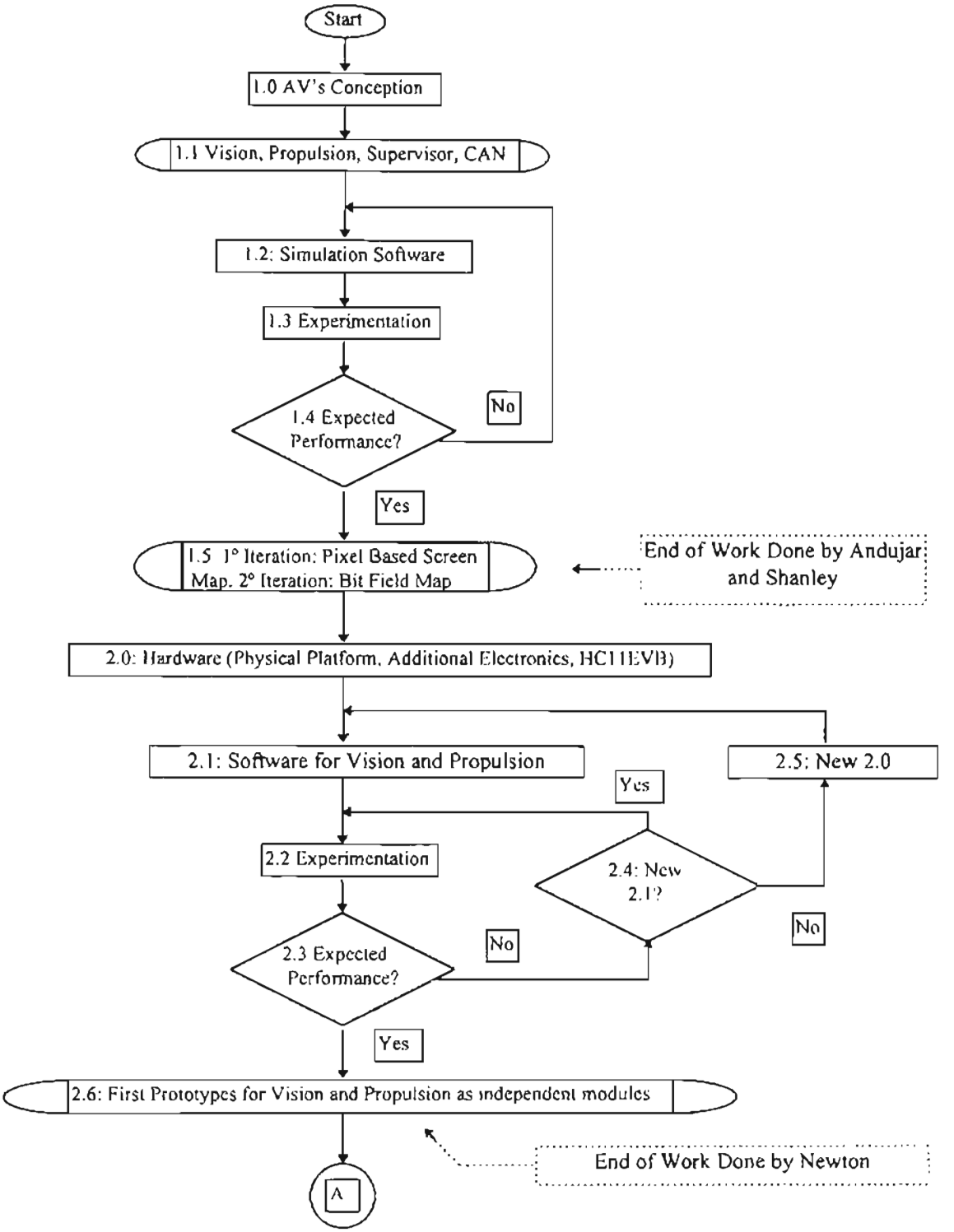


Figure 4a. Scheme for the Sequence of the AV's Development (Work Done by Andujar, Shanley and Newton, Previous to the Present Research Project)

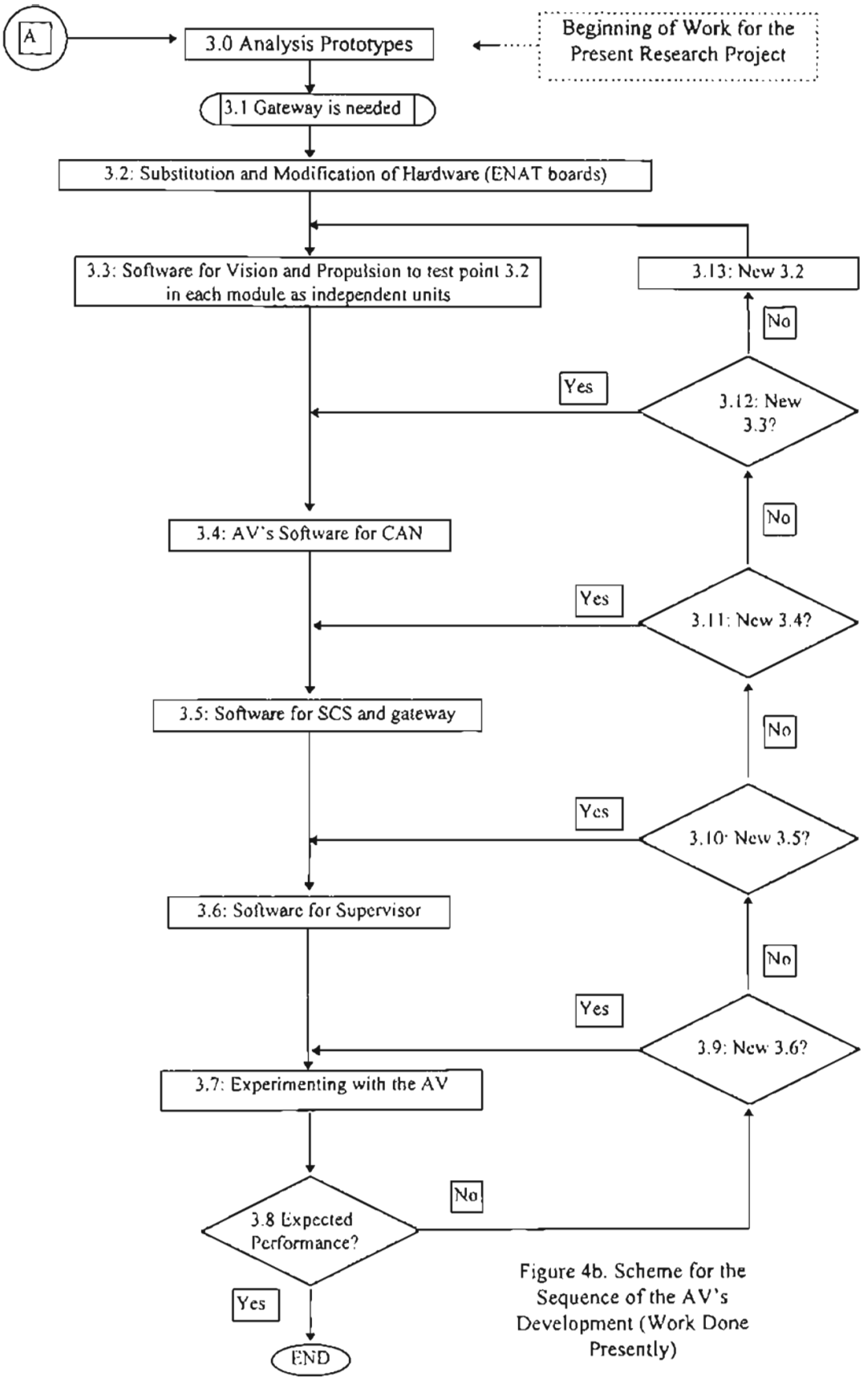


Figure 4b. Scheme for the Sequence of the AV's Development (Work Done Presently)

In other words each box represents a proposal for the design of the subsystem, which after the evaluation of the experiment's results (represented by the rhomboids) is discarded, modified or accepted as a final design. For the most part in this report only the final designs are discussed, unless the intermediate designs offer an interesting contribution by themselves.

The boxes with curved ends show the results of a previous stage in the AV's development.

Figure 4 shows an intrinsic characteristic of the methodology: its cyclic and recurrent nature that implies a process of gradual evolution. Figure 4a. summarizes the previous work done to this research project and shows that by the beginning of the present project a simulation software for the AV, based on the Bit Field Map by Shanley, has been finished as well as the first prototypes for the Vision and Propulsion Modules, which were implemented by Newton in the M68HC11 EVB boards as independent modules unable of performing network communications. Figure 4b. shows the work that corresponds to this present project that begins with the analysis of the prototypes, developed by Newton, with the purpose of substituting the EVB boards by the ENAT boards and ends with the evaluation of the AV's performing of the chosen tasks.

In figure 4b. each one of the boxes (rectangles) corresponds to a proposed design, for a given subsystem, which is actually the result of a methodological process of research as described in figure 5. In this project each one of the five modules of the AV is seen as a subsystem and this is why it is suitable to dedicate for each one of them, one chapter where mainly the final design of the subsystem is presented as well as the findings or problems encountered during the stage of experimentation.

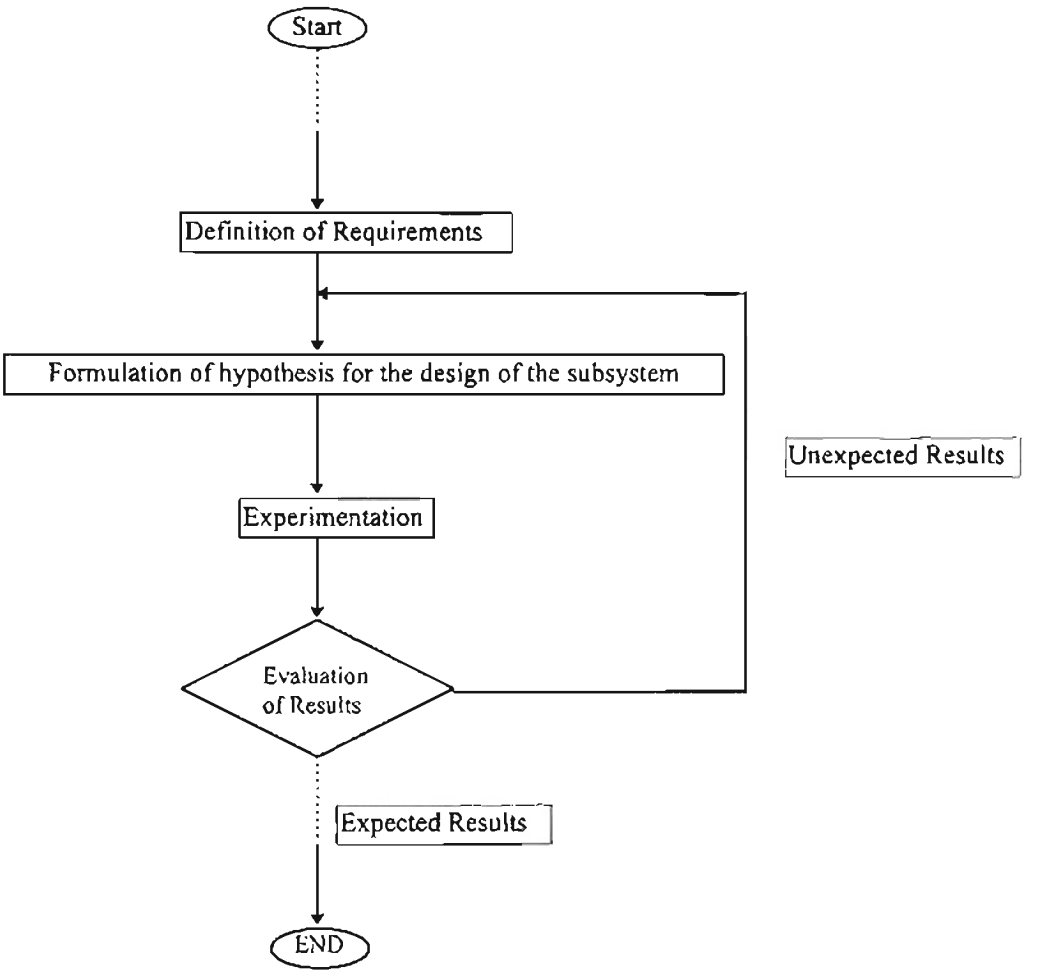


Figure 5. General Scheme of the Methodological Process for the Development of the AV

CHAPTER IV

CONTROLLER AREA NETWORK (CAN)

Following the SAE J1939

To fully define an SAE J1939 network it is required a minimum of seven documents that correspond to the seven layers of communication defined by the OSI model and they are: Physical Layer, Data Link Layer, Network Layer, Transport Layer, Session Layer, Presentation Layer and Application Layer [21]. Altough there is a J1939 document allocated to each layer, not all of them are explicitly defined within J1939. The other layers perform secondary functions described elsewhere. The Physical Layer describes the electrical media (wire or bus). The Data Link Layer provides the reliable transfer of information across the physical link by describing the protocol or rules for constructing a message, accessing the bus and detecting transmission errors.The Applications Layer defines the specific data contained within each message sent across the network.

This project is mainly based on the particular document J1939/21 that describes a Data Link Layer using the CAN protocol and the specification used is the "CAN Specification 2.0 Part B" (Sept. 1991) [22].

The CAN 2.0 B specifies two messages formats, the Standard Frame and the Extended Frame. The SAE J1939 devices must use the Extended Frame format shown in figure 6, that contains 29 identifier bits in the arbitration field. The CAN Extended Frame message will contain a single Protocol Data Unit (PDU). The Application and/or Network Layer provide a string of information that is incorporated into a PDU. The PDU provides a framework for organizing the information of the CAN data frame.

The SAE J1939 protocol data unit or PDU consists of seven fields: priority, reserved, data page, PDU format, PDU specific, source address and data fields (figure 7). Some of the CAN data frame bits

have been left out of the PDU definition because they are controlled entirely by the CAN specification (in other words, the Intel 82527 of the ENAT board manages these bits automaticly) and are invisible to all of the OSI layers above the Data Link Layer. They are: start of frame bit (SOF), substitute remote request (SRR), remote transmission request bit (RTR), identifier extension bit (IDE), cyclical redundancy code error code (CRC), acknowledge field (ACK), and end of frame (EOF).

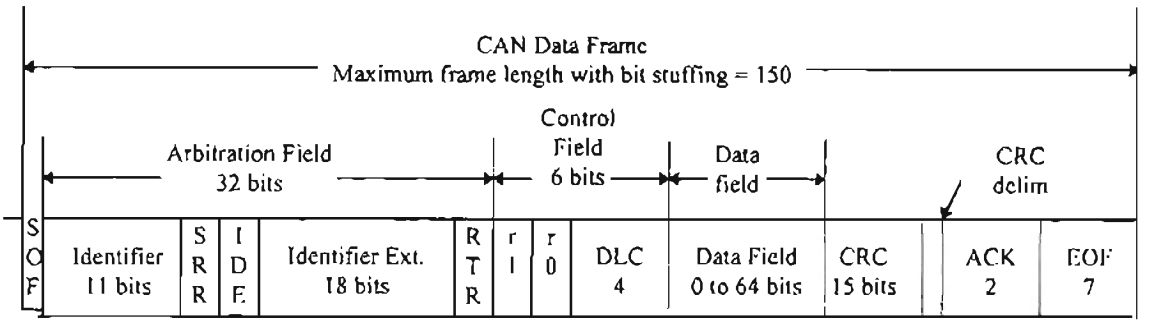


Figure 6. CAN Data Frame

J1939 PDU

	Priority,	R,	DP,	PF,	PS,	SA	Data Field
Bits ->	...3...,	.1.,	.1.,	...8...,	...8...,	...8...,	...0-64...

Figure 7. Protocol Data Unit (PDU)

To identify a Parameter Group in the data field 24 bits are used. The Parameter Group Number (PGN) is a 24 bit value consisting of: reserved bit, data page bit, PDU format and Group Extension fields. A PGN can represent one or more parameters, where a parameter is a piece of data such as engine rpm. Next the fields of the PDU will be defined.

Priority (P): The priority of any message can be set from highest, 0, to lowest, 7. Default for all control oriented messages is 3 and for all other informational, proprietary, request and ACK (acknowledgment) messages is 6.

Reserved bit (R): This bit is currently reserved for future use by the SAE and all messages should set it to zero on transmit.

Data Page (DP): This bit selects an auxiliary page of Parameter Group descriptions. In this project it is set to zero.

PDU Format (PF): This 8-bit field determines the PDU format and is used to determine the PGN assigned to the data field. PGNs will be used to identify commands, data, some requests, acknowledgments and negative acknowledgments.

PDU Specific (PS): Its definition depends on PDU format. If PF is below 240 then PS is a destination address; if PF is 240 to 255 then PS contains a Group Extension:

Destination Address (DA): Defines the specific address to which the message is being sent.

Group Extension (GE): This field in conjunction with the 4 least significant bits of the PF provide for 4096 Parameter Groups per data page.

Source Address (SA): There shall only be one device on the network with a given source address. Therefore the SA assures that the CAN identifier will be unique as required by CAN.

Data Field: When 8 or less bytes are required for expressing a given Parameter Group then all 8 data bytes of the CAN data frame can be used.

J1939 defines two PDU formats: PDU1 Format (PS= Destination Address) and PDU2 Format (PS=Group Extension). PDU1 allows for direction of the CAN Data Frame to a specific destination address (device), while PDU2 can only communicate CAN Data Frames that are not destination specific.

In this project only PDU1 is used. Also the PFs used here begin at 238 and decrement as it is advised in section 3.3 of the J1939/21. The PDU1s used by the AV's modules in this project are presented and described in each module's chapter on the section dedicated to the CAN. At this point it can be said that in this research project some of the PGNs are redefined as shown in the corresponding chapters. They are the PGNs with PFs 238, 237, 236, 235, and 233. The Request and Acknowledgment PGNs with PFs 234 and 232 respectively, are used as defined by the J1939. The meaning of the five PGNs mentioned above is the only aspect of the J1939 that is not followed on this project; in other words, the specifications of the Data Link Layer are always followed completely, but the Applications Layer's are not.

The reasons for this change are explained next.

1) The six sonar readings processed by the Vision Module are specific data to be transferred within the CAN Data Frame and are not standard data defined in the Applications Layer.

2) As explained in Chapter I in the future it is possible to install onto the CAN a speech processing unit and a "robot arm" moving unit. For these applications there is no specification or PGN definition in the J1939. So for the addition of the new messages required by these new modules PF numbers may be assigned starting in 231 and will continue in a decrement way.

3) The Data Link Layer makes some provisions for groups of special functions (like proprietary functions) and defines the Proprietary PGNs as special types of parameter groups. However Proprietary PGNs (directed to users that require their own defined PGNs) are not used, because the messages used in this project cannot meet the constraint of the 2% or less of the vehicle network utilization.

Hardware for the CAN

(the Intel 82527)

According to its manual [11] the 82527 has the capability to transmit, receive and perform message filtering on extended message frames with a 29-bit identifier. All communication or message objects are stored in the RAM of the CAN chip for each node. A transmitting node broadcasts its message to all other nodes on the network. A message is received or not according to the decision of an acceptance filter at each node. The acceptance of a message occurs only if a message object with the same message identifier has been stablished in the CAN RAM for that node. This RAM can be addressed by the CAN controller and the CPU (or M68HC11 in the DNAT board). The CPU controls the registers and bit fields in the RAM.

There are 15 message objects and each one of them consists of the following 15 8-bit registers, as labeled in this project:

- CONT0 and CONT1 that are the control registers to handle transmit and receive related functions.

- ARB0, ARB1, ARB2 and ARB3 or arbitration registers where the 29-bit identifier is stored. This identifier corresponds to the P, R, DP, PF, PS and SA fields of the PDU. If a word of 32 bits is formed with ARB0 as the most significant byte and ARB3 as the least significant one then the identifier can occupy the 29 most significant bits, the least three significant bits of ARB3 are set to zero.

- MCONF or message configuration register.

- DATA0 to DATA7 or data bytes.

In the next section an example of the software developed for the CAN in this project is presented and with this some other aspects of the 82527 will be clear.

Software for the CAN

A general structure to support the CAN in programs written for the ENAT board is developed in this research and presented next. The specific aspects and details are found in the appendixes A, B, C and D. The order must be respected. The code consists of external files to be included and functions or routines written in Small C and HC11 assembly language [18] .

1) Defintion of starting addresses.

2) Including of file CAN_INI.C (with routines in assembler for initialization of CAN, namely the 82527).

3) Register bank mapping for HC11 and 82527.

4) Including other files of interest for the specific application.

5) Global variables.

6) Interrupt Service Routines (with the service for IRQ, which is a routine to handle the message that is received, based on J1939).

7) main()

7.1) Section of Initialization of CAN (in assembler, call of routines defined in CAN_INI.C).

7.2) Call of function "set_mes#()" to set the message object # (all PDU fields are set, except data fields).

7.3) Definition of starting addresses for interrupt routines.

7.4) Interrupts enabling.

7.5) Message or response received flag is reset.

7.6) Main while loop (with the main process).

7.6.1) Call of function "upd#()" to update the data of message object #.

7.6.2) Call of function "send(MESS#)" to send or transmit the updated message MESS#.

7.6.3) Wait while message or response received flag is not reset.

7.7) End of main while loop.

8) End of main().

9) "set_mes#" functions.

10) "upd#()" functions.

11) "send(MESS#)" function.

12) Other functions.

To explain the parts of the general stucture related directly to the CAN some segments of the code for the Gateway (appendix D) are used as an example. First of all the messages to be transferred through the CAN need to be defined:

- A command from the gateway (that is actually a Supervisor's command) to the Propulsion Module. This is a message to be transmitted by the gateway with desired speed ("des_vel"), steering angle ("des_phi"), and the emergency stop ("mot_onoff"); which are all unsigned integers of two bytes.

- An acknowledgment. This is a message to be received by the gateway and sent by the Propulsion Module acknwledging the reception of the command sent first by the gateway.

The addresses are: gateway 0 and Propulsion Module 1.

TABLE 1.

EXAMPLE OF CAN MESSAGES

M.#	P	R	DP	PF	PS	SA	Data fields (8 bytes)							
1	3	0	0	238	1	0	des_vel		des_phi		mot_onoff		XX	XX
2	6	0	0	232	0	1	0	255	255	255	255	0	238	1

M. #. 1: Command; 2: Acknowledgment

After shifting the 29-bit identifier three places to the left the identifiers in hexadecimal results in:

TABLE 2.

IDENTIFIER ASSIGNMENT TO ARBITRATION REGISTERS

Message	ARB0	ARB1	ARB2	ARB3
Command	0xC7	0x70	0x08	0x00
Acknowldgment	0xC7	0x40	0x00	0x08

A message object of the 82527 must first be set in order to define whether it is to be transmitted or it is going to receive a message. So in this example the two "set_mes#()" functions are:

```
/* set message objetct #1 for receiving ACK of command (with desired
        speed and steering angle) from Navig  */

set_mes1() {
        pokeb(MESS1+CONT0,0x55);
        pokeb(MESS1+CONT1,0x55);
        pokeb(MESS1+ARB0,0xC7);
        pokeb(MESS1+ARB1,0x40);
        pokeb(MESS1+ARB2,0x00);
        pokeb(MESS1+ARB3,0x08);
        pokeb(MESS1+MCONF,0x84);        /* to define direction - receive */
        pokeb(MESS1+CONT0,0x99);        /* to make the message object valid and to generate an interrupt
when
                                        the ACK message is received */
}

/* set message object #6 for transmitting command with desired speed and
        steering angle to Navigation */
```

```
set_mes6() {
        pokeb(MESS6+CONT0,0x55);
        pokeb(MESS6+CONT1,0x59);
        pokeb(MESS6+ARB0,0xC7);
        pokeb(MESS6+ARB1,0x70);
        pokeb(MESS6+ARB2,0x08);
        pokeb(MESS6+ARB3,0x00);
        pokeb(MESS6+MCONF,0x8C);        /* to define direction = transmit */
        pokeb(MESS6+CONT0,0xBF);        /* to make the message object valid */
}
```

Now that the messages are set their processing can be done. In the case of the command before it is sent its data must be updated:

```
/* Update data in message object 6
*/
upd6() {

        pokeb(MESS6+CONT1,0xFA);  /*START OF UPDATING */

        poke(MESS6+DATA0,des_vel); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        poke(MESS6+DATA2,des_phi);
        poke(MESS6+DATA4,mot_onoff);

        pokeb(MESS6+CONT1,0xF7);    /*END OF DATA UPDATING */
}
```

Now the command can be sent (mess = MESS6).

```
/* MESSAGE OBJECT SENDS ITS DATA FRAME
*/
send(mess)
unsigned int mess;
{
pokeb(mess+CONT1,0xE6); /* the transmission of this message is requested */
res_rec=1;              /* message or response received flag is set */
}
```

In the case of the acknowledgment, when this or any other message is received by the CAN chip, this device generates a signal that is the input to the IRQ (Interrupt Request) pin of the HC11 and makes the HC11 execute the routine of service for the IRQ interrupt:

```
/* Receive routine: Here the typical receive routine recommended by the
SAE J1939 for Data Link Layer is implemented partially, because only
specific requests and PDU1 format (DA specific) are used */

interrupt irq()
{
d_int();                                        /* disable interrupts */
mess = MES_BASE + (peekb(INT_REG)-3)*16;
if (peekb(INT_REG)>0x02) {
```

```
                        pokeb(mess+CONT1,0xFD); /* clear NewDat */
/*                      PF = (peek(mess+ARB0) >> 3) && 0x00FF;
                        DA = (peek(mess+ARB1) >> 3) && 0x00FF;
                        SA = (peek(mess+ARB2) >> 3) && 0x00FF;
                        if (PF < 240) {
*/                      switch (mess) {
                                case 0xCA10:  /* MESS1 */
                                        for(i=0;i<7;i++)
                                                d[i]=peekb(MESS1+DATA0+i);
                                        res_rec=0;
                                        break;
                        ....
                        ....
                        }
        pokeb(mess+CONT0,0xFD);     /* to indicate that interrupt is not pending */
        e_int();                    /* to enable the interrupts */
}
```

MES_BASE correspond to the address of the first message object and INT_REG is the interrupt register of the 82527, that indicates the source of the interrupt or the message object where the recently received message is stored. In the array d[] the data field of the Ack. message is stored to be processed later on.

CHAPTER V

THE VISION MODULE

Definition of the Module

The Vision Module consists of three parts: the controller, the sensors (sonars) and CAN interface. This module has the purpose of coordinating the acquisition of vision information from the sonar units and sending this data to the supervisor for further processing through the CAN. In this way the vision handles any requests from the supervisor. There are six sonars placed strategically around the wheelchair.

Hardware of the Vision Module

Previous to this project Newton had installed the six sonars on the wheelchair so that their signals were handled by the M68HC11EVB, which would send the sonars' distances to a terminal.

Next it is presented a description of the module's hardware after the process of modification of the module required by the substitution of the M68HC11EVB by the ENAT board. The ENAT board offers a major disadvantage: not all the I/O port pins of the HC11 are available and some of the ones that are available for the user external connection are already used or connected to other elements to perform a specific function of the ENAT board. Since in the HC11EVB all ports were available to the user other port pins had to be used instead of the ones used by Newton [19]; and the corresponding changes in software had to be made.

It was necessary to make two expansions in the ENAT board in order to have the I/O ports available: two 40-pin headers were installed in the slots JP2 and JP3. Table 3 shows the new controller's pin interface including the JP pin and its corresponding connector pin.

TABLE 3

VISION ENAT CONTROLLER PIN INTERFACE

Port Pin	JP Pin	Cone. Pin	Function
PB0	J3,40	19	Transmits pulse from center front sonar (1)
PB1	J3,38	18	Transmits pulse from front left sonar (2)
PB2	J3,36	17	Transmits pulse from front right sonar (3)
PB3	J3,34	16	Transmits pulse from left side sonar (4)
PB4	J3,32	15	Transmits pulse from right side sonar (5)
PB5	J3,30	14	Transmits pulse from rear sonar (6)
PE0	J2,23	12	Polls for the echo signal from sonar 1
PE1	J2,21	11	Polls for the echo signal from sonar 2
PE2	J2,19	10	Polls for the echo signal from sonar 3
PE3	J2,17	9	Polls for the echo signal from sonar 4
PE4	J2,15	8	Polls for the echo signal from sonar 5
PE5	J2,13	7	Polls for the echo signal from sonar 6

The six sonar units have been placed strategically around the wheelchair. The specifications of sonars are in the table 8 of Chapter IX. It has been assumed [19], that the AV will be traveling in the forward direction most of the time. Therefore three sonar units send signals in front of the chair, two sonar units do it outward from each side of the chair, and one sonar unit do it behind the chair. Figure 9 shows the wheelchair with each sonar's signal illustrated.

Figure 8 displays a schematic of the sonar circuitry. In this way, each one of the sonars is initiated by a 5 volt pulse transmitted from the corresponding PORT B's pin on the vision controller. When the sonar unit detects an echo, it pulses the echo pin, which is received in the corresponding PORT E's pin by polling in the vision controller.

The controller is connected to the network, CAN, through a 9-wire cable and a 9-pin male connector plugged into the CAN bus connector of the 82527.

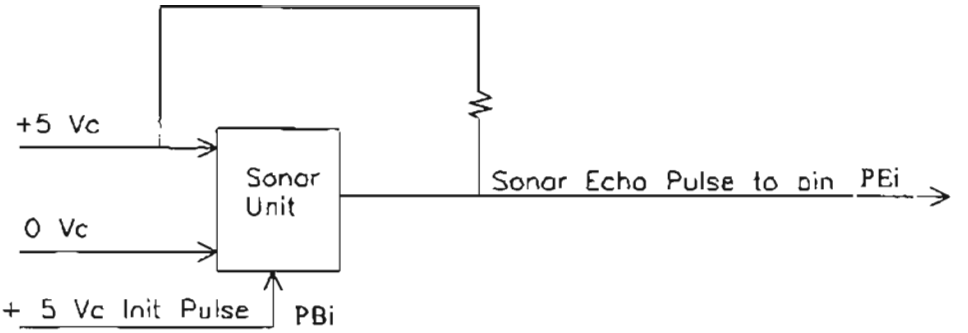


Figure 8. Schematic of the Sonar Circuitry [19, modified]

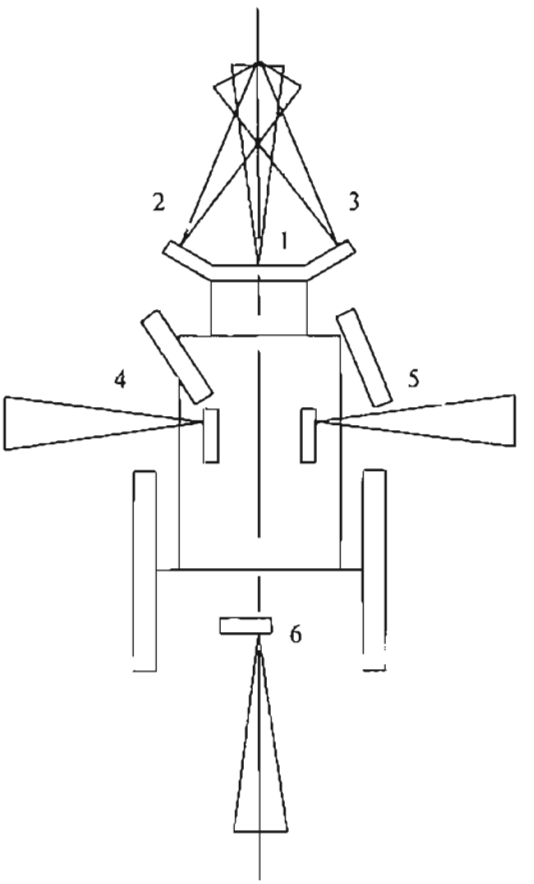


Figure 9. Propulsion / Vision Layout, the number assigned to each sonar unit is shown [19, modified]
(Top View: 1/16 Scale)

CAN implementation

In the vision module the CAN is implemented by following the general structure presented in Chapter IV and developing the corresponding functions. To develop these functions the first step is to define the characteristics of the data to be transferred through the CAN.

The Supervisor, with address 0, makes the request of the six sonar readings and the Vision Module, with address 2, must answer by sending the six readings. Each sonar reading or distance is of type unsigned integer of two bytes and since each message can support only 8 bytes, only four sonar readings, then it is necessary to use twice the Request PGN CAN message transmitted by the Supervisor to request all the six distances: first the readings from Sonar 1 to Sonar 4 and then the readings of sonars 5 and 6. Also there must be two messages to be transmitted from the Vision Module: one with the sonar readings of Sonar 1 to Sonar 4 and the other with the readings of sonars 5 and 6.

The PGNs have the following three-byte structure, in hexadecimal: 0x00 PF 0x00. So it is enough to define the PFs to identify the message's data. The PFs are defined in Small C as:

```
#define Son1to 0xEB     /* 235 */
#define Son56 0xE9      /* 233 */
```

TABLE 4

VISION MODULE'S CAN MESSAGES

M #	P	R	DP	PF	PS	SA	Data fields (8 bytes)							
1	6	0	0	234	2	0	0	235	0	X	X	X	X	X
2	6	0	0	235	0	2	dist1		dist2		dist3		dist4	
3	6	0	0	234	2	0	0	233	0	X	X	X	X	X
4	6	0	0	233	0	2	dist5		dist6		X	X	X	X

M #. 1: Supervisor's Request of Son1to4 2: Vision's Response with Son1to4 information
3: Supervisor's Request of Son56 4: Vision's Response with Son56 information

Software for the Vision Module

The original code for the vision control algorithm done by Newton has the function of producing as only output the distances in centimeters from the sonars to a detected object.). In the main loop the process consists of polling for the echo signal that begins when a sonar unit sends its signal. The sampling frequency of the loop is approximately 8.5 Khz. In the present project this code is modified so it can follow the program's general structure to support the CAN presented in Chapter IV (the new Vision code is shown in appendix B).

The sonars return only one piece of information, the distance to the obstacle. This distance does not mean there is always an obstacle at this location. A maximum range of 270 cm is defined, in this project, for the sonars and when there are no obstacles within this range, that maximum distance is returned by the Vision to the Supervisor Module. After the substitution of the HC11EVB by the ENAT board the readings of the sonars had to be readjusted. Newton [19] calculated the distance by approximation to one straight line by relating the real distance to the count of a software counter (that increments in a loop from zero while waiting for the echo signal). Newton also showed that with his code as the measured distance increases the error respect to real distance also increases. Now in this final design this problem is minimized. A similar process is followed, however instead of making the approximation to one line, it is made to several lines. The results are shown in figure 10, where the readings (measured sonar distances) from sonar unit 1 to an object were used to produce the plot by placing the object at different distances (actual distances).

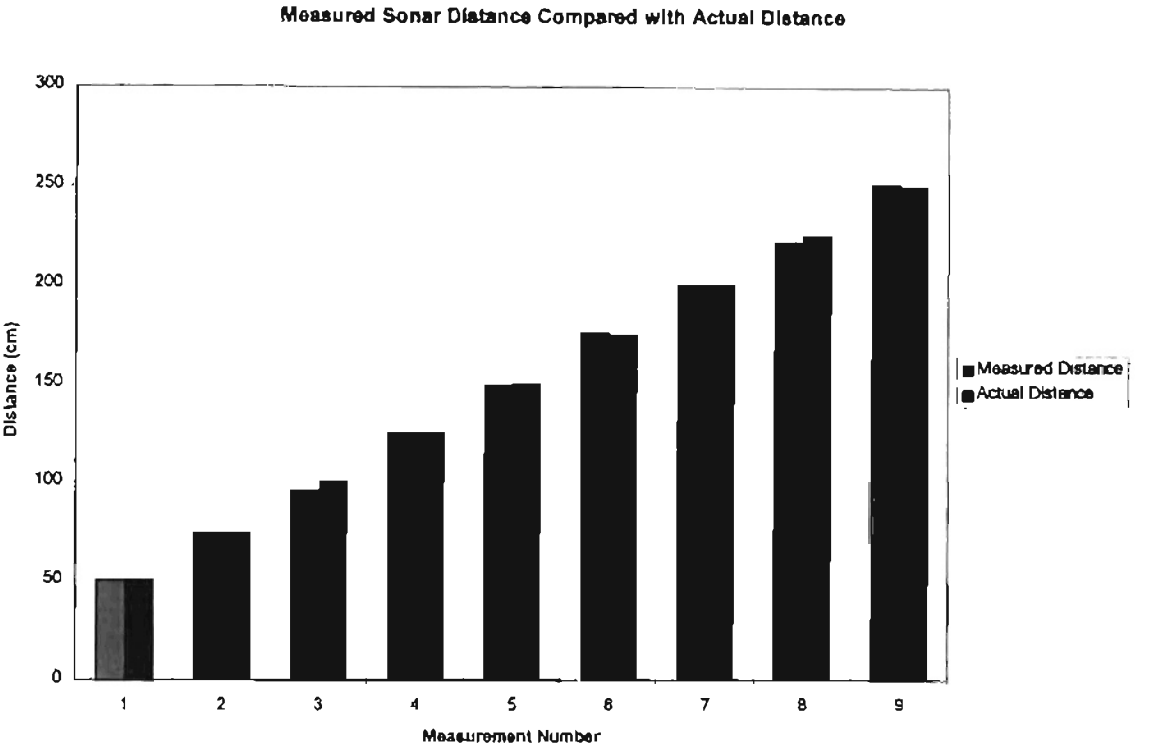


Figure 10. Measured Sonar Distance Compared With Actual Distance

Problems encountered during the Experimentation

The problem discussed next was not found until the actual experimenting with the AV as a whole system. This is a major problem that appears whenever the DC motors of the wheelchair begin to move and the sonars' readings drop and continue to fluctuate. In other words the Vision Module only works fine as long as the motors are off and this problem was not detected in previous stages of the methodology, because both Vision and Propulsion Modules were always tested independently and separately and in this way they worked well. While looking for solutions for the problem the following actions were executed:

1) An independent 12 V power supply was used to feed only the vision controller and 5 V were taken from it to feed only one sonar. The motors were driven by the black box's joystick (the one that comes originally with the wheelchair) and there is always a drop. For instance if an object was put at 127

cm from the sonar then as soon as the motors move the reading drops to 30 cm. With this test it seems that the cause of the problem is not the power supply of the wheelchair.

2) The box where the sonar is placed was wrapped in aluminum foil to avoid any electrostatic effect. However the problem continued.

3) The adjustable elements VR1 and L1 of the sonar unit [20] were readjusted as follows:

VR1: clockwise as far as possible.

L1: clockwise as far as possible and then 3 full turns counterclockwise.

But this actions did not solve the problem.

4) 1) and 2) were done while the sonar box was in contact with the wheelchair's metal structure. As soon as the sonar box is removed and placed in a separate place, like a wood stick, then the problem disappears. However if the wood piece, where the sonar box lies, is put on the metal structure the problem appears. This suggests that vibrations may be affecting, but then the wood piece with the sonar box is moved to different positions, not touching the wheelchair, and it is found that in some positions the problem appears, but as it is placed further the problem begins to disappear until it is gone completely.

After all the above it seems that the cause of the problem consists mainly of an electromagnetic interference. The solution is to move the sonars away from the wheelchair's motors as far as each sonar demand. This solution is implemented partially, because of time constraints of this project; and only the three sonar units of the front are placed in an adequate position further to the front of the wheelchair (approximately 8.5 cm from its original position). The other sonars will have to be moved in a future project.

CHAPTER VI

THE PROPULSION MODULE

Definition of the Module

The Propulsion Module consists of four parts: the controller, the CAN interface, the sensors and the electronics section for the module's output. The Propulsion Module has the purpose of coordinating the acquisition of speed and position information from the sensors (tachometers and encoders respectively) and sending these data to the Supervisor for further processing through the CAN. Also the Propulsion Module receives desired velocity and steering angle through the CAN and has to produce the control action to achieved such desired values. Previous to the present project Newton had worked in the development of the Propulsion Module as an independent unit. A summary of the stages of his work is presented as follows:

1) Definition of the dynamics of the system (wheelchair) through basic kinematics relationships.

2) Substitution of electrical signals and need of tachometer and encoder to measure wheel speed and displacement.

3) System identification, obtaining differential equations that relate inputs to outputs.

4) Controller design: two PI controllers were designed and implemented to control average velocity and rate of rotation, because of their effectiveness in DC applications (a derivative term was not implemented for reasons discussed by Newton).

Hardware of the Propulsion Module

Previous to this project Newton had developed a first prototype for the Propulsion Module that used a M68HC11EVB to control the movement of the wheelchair by receiving desired speed and steering angle as voltages produced from a PC joystick.

Next it is presented a description of the module's hardware after the process of modification of the module required for the substitution of the HC11EVB by the ENAT board. As explained before the ENAT board offers a major disadvantage: not all the I/O port pins of the HC11 are available and some of the ones that are available for the user external connection are already used or connected to other elements to perform a specific function of the ENAT board. Since in the HC11EVB all ports were available to the user

TABLE 5

ENAT PROPULSION CONTROLLER PIN INTERFACE

Port Pin	JP Pin	Cone Pin	Function
PB0	J3,40	19	Port B output bit 0 for rate of rotation and average velocity
PB1	J3,38	18	Port B output bit 1 for rate of rotation and average velocity
PB2	J3,36	17	Port B output bit 2 for rate of rotation and average velocity
PB3	J3,34	16	Port B output bit 3 for rate of rotation and average velocity
PB4	J3,32	15	Port B output bit 4 for rate of rotation and average velocity
PB5	J3,30	14	Port B output bit 5 for rate of rotation and average velocity
PB6	J3,28	13	Port B output bit 6 for rate of rotation and average velocity
PB7	J3,26	12	Port B output bit 7 for rate of rotation and average velocity
PA1	J2,25	30	Captures right wheel position data
PA2	J2,27	31	Captures left wheel position data
PA4	J2,31	33	Loads average velocity data to corresponding register in demultiplexing section
PA5	J2,33	34	Loads rate of rotation data to corresponding register in demultiplexing section
PA6	J2,35	35	Governs the relay which connects and disconnects the D/A converters from the wheelchair's electronics
PC5	J3,14	6	"Emergency Stop" that tells the computer to connect or disconnect the D/A converters from the wheelchair's electronics.
PE4	J2,15	25	Captures right wheel velocity from the right tachometer
PE5	J2,13	24	Captures left wheel velocity from the left tachometer
PE6	J2,11	23	Connection made (used only for point 3.3 of methodology)
PE7	J2,9	22	Connection made (used only for point 3.3 of methodology)

other port pins had to be used instead of the ones used by Newton [19]; and the corresponding changes in software had to be made and the following actions were performed:

1) It was necessary to make two expansions in the ENAT board in order to have the I/O ports available: Two 40-pin headers were installed in the slots JP2 and JP3. Table 5 shows the new controller's pin interface, indicating the JP pin and the corresponding connector pin.

2) The original connections of the input capture pins, PA1 and PA2, to other elements in the ENAT board were removed in order to enable the pins completely for capturing right and left wheel position data from two encoders respectively (details of these changes in diagrams in appendix G).

3) Newton had used all pins of PORT B and PORT C to output the values of average velocity and rate of rotation respectively, however, since only three pins of PORT C can be used in the ENAT board, the new section for demultiplexing the data shown in figure 11 was designed and implemented (the corresponding modifications were also done to the software).

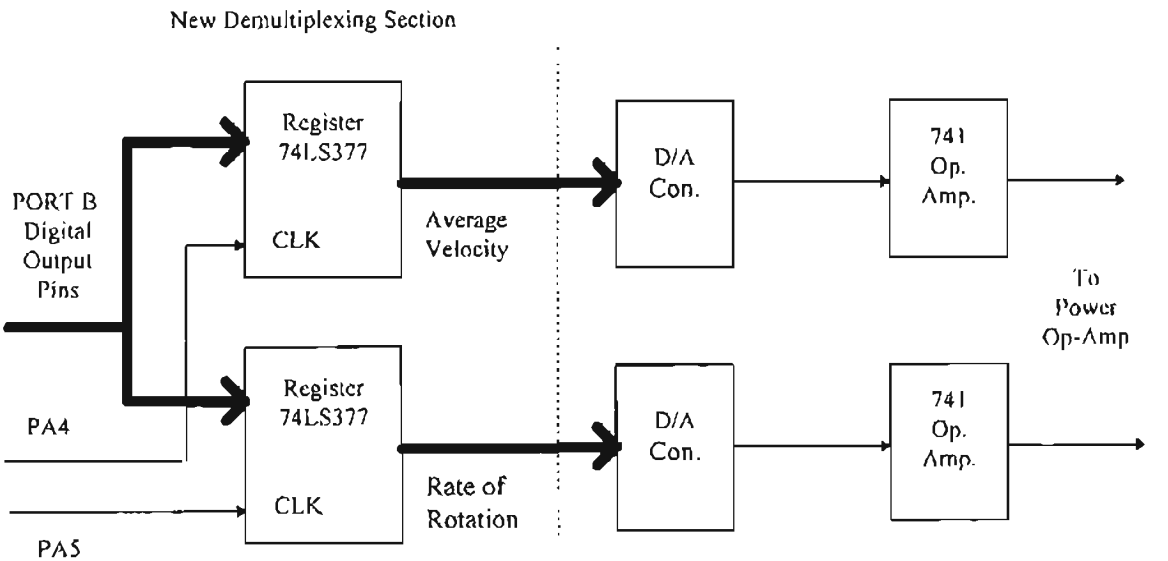


Figure 11. New electronics section of the Propulsion Module's output

A tachometer and an encoder were installed by Newton above each of the two DC motors that drive the chair. The tachometer and encoder are powered from a small flat belt that runs directly from each

DC motor's gearbox that moves each drive wheel. The specifications of these elements are found in table 8 in Chapter IX.

At the end of his work Newton leaves two potentiometers, contained in a PC joystick, to provide the controller with the desired velocity and steering angle. In the present project connections are redone to use the PC joystick only with the purpose of having an auxiliary tool mainly to develop the Vision and Propulsion modules as independent units. Later the desired velocity and steering angle are delivered from the Supervisor through the CAN and the PC joystick is not used anymore.

Newton provided a motor ON/OFF signal through input pin PA0 that was activated by a button of the PC joystick. This button was used in the present project (although now connected to pin PC 5) to provide an emergency stop temporarily, to experiment and study the performance of the AV, since the cable of the PC joystick was long enough to carry out such tasks during the first trials. However, this PC joystick's button was later replaced by the emergency stop button located at the top of the computer box. This input works also along with a motor ON/OFF status or command sent by the Supervisor through the CAN to start or stop the motors.

The controller is connected to the network (CAN) in the same way as the Vision Module.

CAN implementation

In the propulsion module the CAN is implemented by following the program's general structure presented in Chapter IV and developing the corresponding functions. To develop these functions the first step is to define the characteristics of the data to be transferred through the CAN. All data are of type unsigned integers of two bytes.

The Supervisor sends the command message with the desired speed, steering angle and motor ON/OFF status. Once this command is received by the Propulsion Module then this sends an acknowledgment message to the Supervisor to let it know that the command arrived well. Also the Supervisor can request in one message the current speed ("RoadSpeed"), heading angle ("Compass"), and rate of rotation ("t_d" that stands for "theta_dot"). The rate of rotation is then used by the Supervisor to

obtain the current steering angle. These magnitudes are sent in a response message by the Propulsion Module and are shown in figure 12.

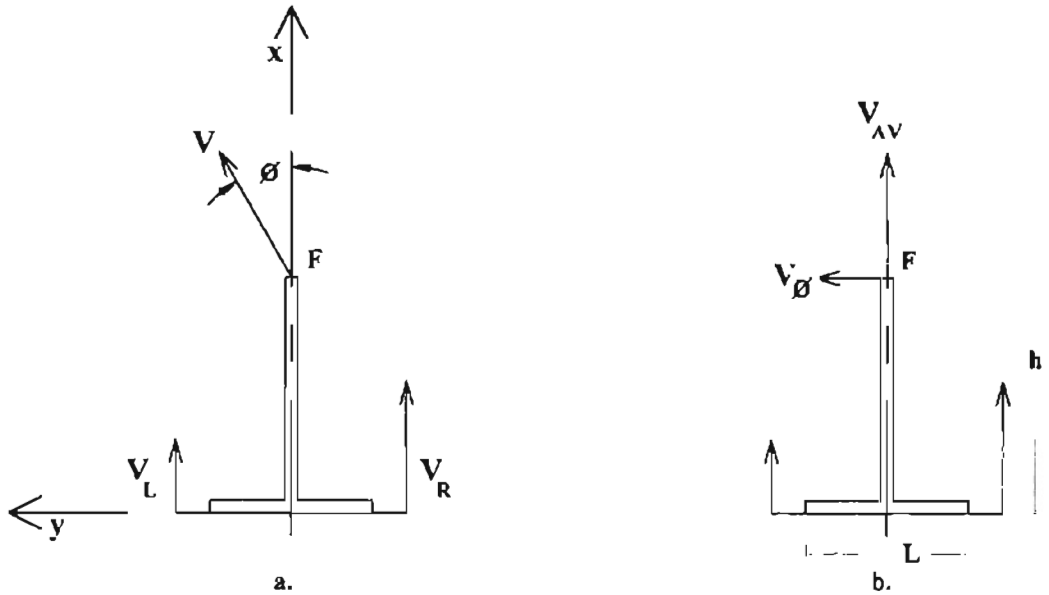


Figure 12. Kinematic Illustration of Velocities [taken from Newton, 19]
(Top View)

The absolute velocity (V) and steering angle (ϕ) are shown in figure 12a at point F. And the velocities of the left and right ends of the rear axle are shown as V_L and V_R. In figure 12b, absolute velocity and steering angle have been resolved into orthogonal components. Average velocity (V_{AV}) is the x component of absolute velocity and (V_ϕ) is the y component. (The equations (1) to (5) were found by Newton and are used here to obtain the equation (6) that defines the steering angle to be used by the Supervisor).

(1) $$V_{AV} = V \cos\phi$$

(2) $$V_\phi = V \sin\phi$$

V_{AV} is also:

(3) $$V_{AV} = \frac{V_R + V_L}{2}$$

$\dot{\theta}$ is the rate of rotation of the wheelchair such that the heading angle of the chair (θ) is measured from the positive x axis. Putting $\dot{\theta}$ in terms V_L and V_R gives:

(4) $$\dot{\theta} = \frac{2\,(V_R - V_L)}{L}$$

and putting $\dot{\theta}$ in terms of V_ϕ gives :

(5) $$\dot{\theta} = \frac{V_{\phi}}{h}$$

Note that the heading angle of the chair is not the same as the steering angle. The steering angle (ϕ) is measured with respect to a coordinate system that is attached to the chair. The heading angle (θ) is measured with respect to a coordinate system that does not move with the chair.

From (1), (2) and (5) it is found that:

(6) $$\phi = \tan^{-1}\left(\frac{\dot{\theta} * h}{V_{AV}}\right)$$

Equation (6) will have to be used by the Supervisor to obtain the current steering angle.

The Supervisor, with address 0, also requests current X and Y position referred to the fixed reference system with origin at the place where AV started to move. These data are sent by the Propulsion Module, with address 1, to the Supervisor through the CAN.

As explained in the CAN section for the Vision Module only the PFs are needed to identify the new data:

TABLE 6

PROPULSION MODULE'S CAN MESSAGES

M #	P	R	DP	PF	PS	SA	Data fields (8 bytes)							
1	3	0	0	238	1	0	des_vel		des_phi		mot	onoff	X	X
2	6	0	0	232	0	1	0	255	255	255	255	0	238	1
3	6	0	0	234	1	0	0	237	0	X	X	X	X	X
4	6	0	0	237	0	1	RoadSpeed		Compass		t_d		X	X
5	6	0	0	234	1	0	0	236	0	X	X	X	X	X
6	6	0	0	236	0	1	Xpos		Ypos		X	X	X	X

M #. 1: Supervisor's Command
2: Propulsion's Acknowledgment
3: Supervisor's Request: RS_C_td
4: Propulsion's Response with RS_C_td
5: Supervisor's Request: Xypos
6: Propulsion's Response with Xypos

```
#define RS_C_td_PF 0xED        /* 237 */
#define XYpos_PF 0xEC          /* 236 */
```

Software for the Propulsion Module

The original code for the propulsion control algorithm done by Newton has a main loop with sampling frequency of 32 Hz. In this loop the process consists of implementing the two PI controllers for speed and steering angle, reading the wheels' speeds, calculating the heading angle and with it the X and Y position. To do this Newton created a look-up table with sine and cosine values is used. The Small C compiler does not handle decimal numbers and unsigned integers are used for all data. So the variables are shifted up by half of 65535. This provides a zero point (offset) from which all variables are measured. Newton [19] wrote a library (in file hc11fp1.c) to perform the arithmetic operations to the mentioned modified numerical data. In the present project this code is modified so it can follow the program's general structure to support the CAN presented in Chapter IV (the new Propulsion code is shown in appendix C).

Problems encountered during the Experimentation

As it happened in the Vision Module's development the following problem was not discovered until an advanced stage of the AV's implementation; the designs of the Serial Communications Section and of the Supervisor. The problem consists of an instability or fluctuation in the readings of heading angle and X-Y position, even when the motors are off. It was not until the design of a "primary supervisor" was being tested that it was possible to see simultaneously on the screen the current speed, heading angle and X-Y position. Because of the limited amount of memory it was not possible to include in the HC11EVB the required functions to display information in the terminal along with all the software for the Propulsion Module. So before point 3.6 of methodology the control of speed and steering angle was always performed well without realizing what was going on with heading angle and X-Y position.

A solution for the problem was not found in this research, but the following actions were carried out.

1) The Input Capture interrupts (from which the X-Y data proceed indirectly) were enabled and the rest of interrupts disabled and they worked fine, although the main while loop was "empty" (it was only a "while (1)"). As soon as something (any line) is added in the while loop the Input Capture interrupts do not work properly.

2) Since many nested functions are used, this causes the stack to grow very fast, so the stack pointer was moved underneath the data and code, but the problem was still present.

3) The interrupts were tested beginning with one and adding one more at a time and having the main while loop empty. Everything worked fine, except when any line was added to the main loop.

4) Global variables were checked to discard possibility of a variable's use by two or more interrupt subroutines.

5) The content of the main loop was transferred to the timer interrupt routine, so it happens every second. The X-Y information is correct, but the control is affected. The frequency was increased to 7 Hz and 15 Hz, but X-Y information is not correct anymore.

Somehow the input capture interrupt subroutine (where a counter counts the pulses from the encoder) yields a higher count than the real one, in other words this interrupt is triggered more often than it really happens. This seems to be a software problem related to the compiler (also because of the while loop added statements problem). However, another reason for the fluctuation in heading angle and X-Y position is the use of a look-up table for cosine and sine and not having the possibility of handling the decimal part of floating point numbers in a more accurate way (these numbers are used in all calculations like some constants to make certain conversions).

CHAPTER VII

THE SERIAL COMMUNICATIONS SECTION AND THE GATEWAY

Introduction to the SCS and the gateway

As explained by Shanley [24] it was decided to install the Supervisor on the motherboard of a Personal Computer 486DX. There are several reasons for this choice. The first is the memory limitations on the HC11 family. As indicated in Chapter I one of the functions of the Supervisor is the mapping of the environment; the map alone will require more memory than is available on the microcontrollers. Nowadays the PCs can have megabytes of memory on the motherboards themselves. A second reason for using the PC is that a hard drive may be added to provide a black box recorder. In this way a detailed description of the operations of the AV would be provided as well as storage for multiple maps of different environments. Besides the AV as a stand alone system would need storage for the source code in case a reboot is needed. The HC11 do not provide disk storage and therefore is not a good choice for the Supervisor.

At this point of the AV's design a problem arised: how was the PC (Supervisor) going to communicate with the ENAT boards (Vision and Propulsion Modules), if the PC does not have a CAN interface? The only means available on the ENAT boards used in this project to communicate with a PC is through its RS-232 serial port. So to solve the problem a way to stablish and implement this communication had to be found.

It was clear that a serial communications system had to be designed between the PC and the ENAT board. The software and hardware from the PC to the ENAT board was called the Serial Communications Section, SCS. This section requires a protocol that adjust to serial communication for RS-232 ports. Now in the AV there are two networks: one that supports the CAN protocol and the other that

supports RS-232 serial communications. According to the document SAE J1939/31, Draft for the Network Layer [21], a gateway is a device that permits data to be transferred between two networks with different protocols. This is why a third ENAT board is needed to act as a gateway.

Hardware for the SCS and Gateway

The ENAT board to install the gateway is provided with a full-duplex asynchronous serial Communications Interface (SCI) that handles a standard NRZ format (one start bit, eight data bits and one stop bit; the word length is defined by the Serial Communications Control Register 1, SCCR1). The DNAT board allows an RS-232 port (appendix G) connected to the SCI to communicate at a 9600 baud rate.

The SCI transmitter and receiver are functionally independent, but use the same data format and baud rate. Data transmission is initiated by a write to the Serial Communications Data Register, SCDR, provided the transmitter is enabled. Data stored in the SCDR is transferred to the transmit serial shift register. When the SCDR is read, it contains the last data byte received, provided that the receiver is enabled.

In regard to the SCS, it is established that the UART I/O address, to be used by the PC, is selected by software depending on the COMM port used. In this project COMM number 2 is used. 8 bits, no parity and stop bit define the word length and the baud rate is 9600. A 25-wire flat ribbon cable of approximately 5 meters of length is connected in one extreme to the PC serial port and its other extreme is connected to a 9-wire flat ribbon cable through a 25-pin to 9-pin adapter. The other extreme of the 9-wire cable is connected to the RS-232 port of the gateway. The long cable will allow the AV to move far enough from the PC. As an autonomous or stand alone system the AV will have the PC's motherboard installed in the same box where the other three computers are; but while the first AV's prototype is being tested the motherboard will be in the PC frame in order to take advantage of the screen and the keyboard.

Software for the SCS and Gateway

In this section two programs are described. One is implemented in the PC for the management of the SCS's resources and is called the Primary Supervisor (that is found in appendix D). The other program consists of the software to handle the gateway functions, it is called the Gateway and is found in appendix D.

The Primary Supervisor and the Gateway work together, they complement each other. This is so because they exchange information related to the same messages. Both programs have to handle all the five messages implemented in the AV and their five corresponding response messages: the two requests messages and their responses handled by the Vision Module (table 4) and the command message and its acknowledgment message and the two request messages and their responses all handled by the Propulsion Module (table 6). To understand fully how one program works it is necessary to know how the other works; so first their structures are described and then their functioning will be explained by an example.

The Primary Supervisor's basic structure is as follows (the lines in {} are optional):

1) "Include files" section.

1.1) Inclusion of file IBMCOM3.C (set of routines for doing low-level serial communications on the IBM PC. It is found in appendix G).

2) Definition of constants

3) Global Variables

4) Function prototypes

5) main()

5.1) Definition of buffers for transmitting, receiving and handling data (bufTX[], bufRX[] and buf[], respectively)

5.2) Call of function "init_port" to indicate the serial port, baud rate and word format.

5.3) Assignment of the character to start the message to transmit (bufTX[0]=0x09).

5.4) Do loop:

5.4.1) Call of function to clear buffers.

5.4.2) Call function to "choose_action" (the choice corresponds to the number of the message to be sent).

5.4.3) Switch statement according to "choice":

5.4.1.1) Case "choice":

- bufTX[1]="choice" (to inidicate message #).

{- function "ascii(byte)" gives the two ASCII characters of the hexadecimal value of "byte"}

- ASCII characters are assigned to bufTX[2]...

- Function "send_command(bufTX[])"transmits the information contained in bufTX to the gateway.

- Function "wait_for_response(bufRX[])" receives the response message in bufRX.

- Function "i_ascii(bufRX[n], bufRX[n+1])" forms a single character by joining the two ASCII characters bufRX[n] and bufRX[n+1] and it is assigned to buf[].

- Other processes.

5.4.4) End of switch statement

5.5) End of Do loop

6) End of main()

7) Functions

The Gateway follows the general structure of Chapter IV (appendix G shows the code), therefore only the different and important aspects of the main while loop are presented next (the lines in {} are optional):

```
while(1)
/* beginning of while loop */
        while(!(SER_in()==0x09));          /* waits for start of message reception */
        i=0;
        while(13!=(bufRX[++i]=SER_in())); /*receives characters until carriage return */
        switch (bufRX[0])                  /* checks number of message */
        /* beginning of switch loop */
                case "bufRX[0]":
                        {- Call of function "i_ascii(bufRX[n],bufRX[n+1])"}
                        - Call of function "upd#()"
                        - Call of function "send(MESS#)"
                        - Wait while message or response received flag is not reset.
                        - SER_out(0x09) /* transmits start of transmission */
                        - SER_out(0x#)  /* sends # of message to transmit */
                        - Call of function "ascii(data)"
                        - Use of SER_out to transmit ASCII characters
                end of case
        end of switch
end of while
```

For example if the user wants to see on the screen the current speed, he/she types "2". The Primary Supervisor knows that this corresponds to the message #2, which is the request of RS C_td whose PGN is 0x00 0xED 0x00. Then "choice" is 2 and "send_command(bufTX)" transmits this PGN in bufTX. So bufTX[0]=0x09 (which is always the start of any message), bufTX[1]=0x02 (message number), bufTX[2]=0x30, bufTX[3]=0x30 (these values correspond to the hexadecimal values in ASCII for the first byte of the PGN, 0x00; the ASCII value for zero is 0x30). And so on. As a final character "send_command" sends a carriage return, to indicate end of message.

When the Gateway gets 0x09, start of message, then by using SER_in() (which is a new Small C library function created by New Micros, Inc. [18] that manages the SCDR of the DNAT board for reception) bufRX is filled with the ASCII characters until a carriage return appears. Now bufRX[0] has the message number and with it the Gateway finds out what to do. By using "i_ascii" the received ASCII characters are joined together to form the original information. So if bufRX[1]=0x30 and bufRX[2]=0x30, then the first byte of the requested PGN, 0x00, is obtained again and so on.

Now the PGN is sent with "send" through the CAN to the Propulsion Module. The Gateway waits for the Propulsion's response. When it gets to the gateway then "SER_out" (which is the complement of SER_in() for transmission) sends 0x09, start of message; and 0x02, message number.

The current speed or average velocity "a_v", that the gateway just got from the Propulsion, is a variable of type unsigned integer, so it consists of two bytes which are processed in the following way:

```
ascii(a_v>>8);      /* converts the hexadecimal value of the high byte in the two ascii
                       characters */
SER_out(l);         /* transmits left ascii character */
SER_out(r);         /* transmits right ascii character */
```

The process is repeated for "a_v's" low byte. While this is happening "wait_for_response(bufRX)" in the Primary Supervisor is receiving and holding the ascii characters in bufRX. So now with "i_ascii" the real value of "a_v" will be obtained as follows:

```
buf[0] = i_ascii(bufRX[2],bufRX[3]);
buf[1] = i_ascii(bufRX[4],bufRX[5]);
temp = (buf[0]<<8) + buf[1];
ARoadSpeed = (temp - 32767.0)/10;
```

So in buf[0] and buf[1] are the high and low bytes of a_v, which are joined in temp to form the original unsigned int (produced by the Propulsion). Then this value is not only converted to an integer (by subtracting 32767, the offset), but also to a floating point number. It is divided by 10 because it had been rescaled by 10 in the Propulsion. And so ARoadSpeed is the current speed requested by the user.

Problems encountered during the Experimentation

The functions "ascii" and "i_ascii" use the inequalities, and while developing them the following errors of Small C Compiler were found:

when X = Y then X < Y and X > Y are true and X >= Y and X <= Y are false.

The solution to this problem was to "fool" the compiler by adding or subtracting "1" to any of the variables as needed.

CHAPTER VIII

THE SUPERVISOR MODULE

Definition of the Module

The Supervisor Module is the "brain" of the AV that tells it what tasks to perform and how. Essentially it "supervises" the Vision and Propulsion Modules by providing commands for desired speed and steering angle, based on requested sonar information, current speed, heading angle and X-Y position. To send these commands and to request and receive this information the Supervisor uses the CAN through the SCS and the gateway.

Therefore, so far, there are five messages that the Supervisor sends to the CAN and which were already explained in past chapters:

1) Command of desired speed and steering angle (Chapter VI)

2) Request of current speed, heading angle and rate of rotation (Chapter VI)

3) Request of X-Y position (Chapter VI)

4) Request of information of sonars 1 to 4 (Chapter V)

5) Request of information of sonars 5 and 6 (Chapter V)

The hardware (as well as part of the software) for the Supervisor Module was presented in Chapter VII and corresponds to the SCS.

The Scope and Limitations Section of Chapter I explains that this research project is completed with the demonstration of the validity of a supervisor module that allows the AV to perform the following of a moving object and obstacle collision avoidance. This Supervisor is called SUP4.C. Also the same section explains that the development of a different supervisor based on Andujar and Shanley's work is

also explored, it is not completed due to the problems encountered during the experimentation with the Propulsion Module (Chapter VI). The part of the software developed for this Supervisor in this project is called ZZ_SUPR4.C. In this chapter both SUP4.C and ZZ_SUPR4.C are described.

Software for the Supervisor: the ZZ_SUPR4.C

This supervisor is supposed to use both memory map and sensor path planning routines to reach the desired location. This software developed by Andujar and Shanley is a set of files and lots of functions. Since there is no flow-chart of the program sequence and the size of the software and its few comments do not help to understand its structure and follow the sequence of events, the first step is to execute an "analysis and organization of the whole simulation software".

To do this the Shanley's Supervisor that implements Gaussian Fuzzy Sets is used. All files are joined in a big unique text file of, approximately, 5000 lines. The program CFLOW.EXE by Mark Ellington and Larry Steeger [6] is applied to this file to decipher the program structure; in this way with a change in a module, it will be known what will happen to other functions that use it and which ones they are. With the help of CFLOW some new files are made to present the general structure of the software and are presented in appendix F.

Based on this information the ZZ_SUPR4.C begins to be developed. Since it is a simulation the original program uses the screen of the PC mainly as the input environment as well as the keyboard to choose desired targets interactively. In the ZZ_SUPR4.C the screen is not used anymore and instead the input environment comes from memory. Now only one desired target can be chosen per program execution, and it is input by the user and once it is reached the program terminates. Of course this supervisor includes the content of the Primary Supervisor of Chapter VII in a function called CAN_interface(). Since, by the time this software was being developed, the AV was on the bench without possibility to move the vision module was still simulate, but this time taking as input the map loaded in memory. This software could not be tested successfully, because it uses the X-Y position information and

this is not being produced correctly as explained in Chapter VI, however the software is shown in appendix F for its future use

Software for the Supervisor Module: SUP4.C

The program SUP4.C is found in appendix E and consists of two general parts: the Fuzzy Logic Controller, FLC, and the SCS for CAN interface. This second part is implemented by using the Primary Supervisor of Chapter VII as a function called CAN_interface().

The FLC corresponds to a typical fuzzy controller with four modules according to Fowler [8] and shown in figure 13 which are given a specific ordering in this project:

1) Knowledge Repository (where the fuzzy sets and rule base are defined)

2) Fuzzifier

3) Process Logic (inference engine)

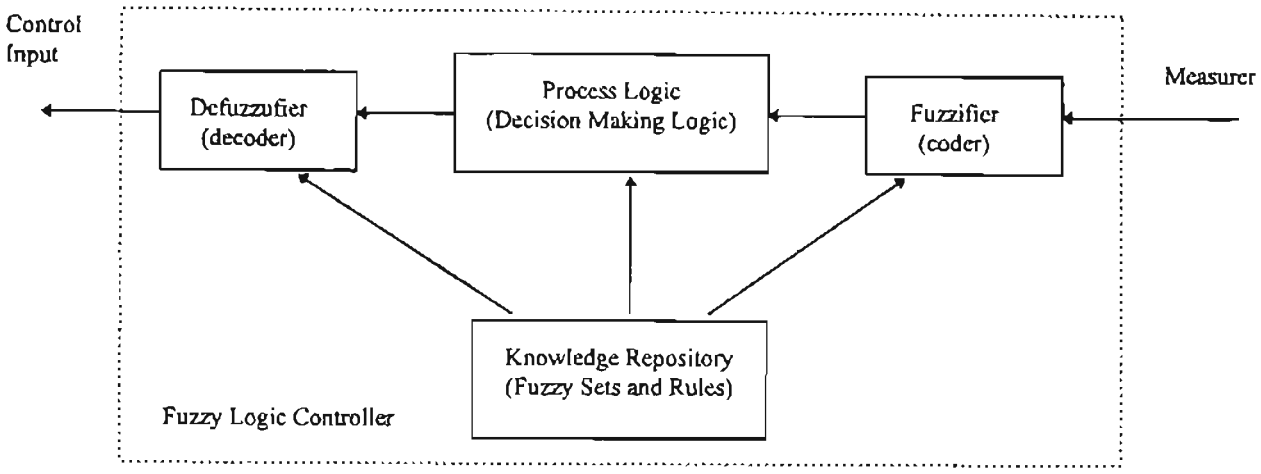
4) Defuzzifier

In figure 13 it is shown the block diagram of the FLC and the code corresponding to that diagram (only typical representative lines of code are shown). This is done so to show how easy it is, with this design of FLC, to identify each module of the fuzzy controller in the code, this makes clearer the design and gives high flexibility for modifying the controller while tuning it.

Knowledge Repository

Fuzzy Sets

The inputs to this FLC of the Supervisor are the relative distance between the sonar 1 and the moving object, named S1 and the relative distance between sonars 2 and 3 and any object (obstacle), labeled S2 and S3 respectively. These distances are found by subtracting a desired distance (to be kept between the sonar and the object) from the readings of, only, the three sonars of the front (the other three




```
/* Crisp inputs */

son1=Range[0]-des_dist1; ...

/* Knowledge Repository
   (Fuzzy Sets)

s1max=40.0; ...

s1c=-s1max;
s1dp=0;
...
s1hsc=s1max; ...

/* (Rule Base) */

rsp[0][0][0]=speedrs; rsp[0][0][1]=speedrs;
rsp[0][0][2]=speedrs; ...

/* fuzzification */

mux1[0]=clo(s1hsc,s1c,son1);
mux1[1]=memfun(s1hsdp,s1dp,son1);
```

```
/* Process Logic */

for (p=0;p<nx1;p++)
        for (q=0;q<nx2;q++)
                for (r=0;r<nx3;r++)

        fi[p][q][r]=mini(mux1[p],mux2[q],mux3[r]);

/* defuzzification */

sumNsp=sumDsp=sumNan=sumDan=0;
for (p=0;p<nx1;p++)
        for (q=0;q<nx2;q++)
                for (r=0;r<nx3;r++)
                {
                if (fi[p][q][r])
                sumNsp = sumNsp + rsp[p][q][r]*
                        fi[p][q][r]/rsps[p][q][r];
                sumDsp = sumDsp +
                        fi[p][q][r]/rsps[p][q][r];
                ...
                }

RoadSpeed = (int)(sumNsp/sumDsp);
```

Figure 13. FLC block diagram and software implementation

could not be used in this project, because of the problems encountered and explained in Chapter V). The outputs of the FLC of the Supervisor are the desired velocity, V, and the desired steering angle, SA.

After defining inputs and outputs to the FLC, then the input and output fuzzy sets can be defined.

Then there are five elements or linguistic variables to deal with: S1, S2, S3, V and SA .

The corresponding physical domains of S1, S2, S3, V and SA are *S1, S2, S3, V* and *SA* respectively. The term sets containing the linguistic values for the five linguistic variables are:

LS1 = {C, DP, F}

LS2 = *LS3* = {VC, C, F}

LV = {RM, RS, Z, FS, FM}

LSA = {R, ST, L}

So for example μ_{LS1} = DP or μ_{LV} = FS. Table 7 shows the meaning of the linguistic values.

TABLE 7

MEANING OF LINGUISTIC VALUES FOR FLC

Name	Meaning
VC	Very Close
C	Close
DP	Desired Position
F	Far
RM	Reverse Medium
RS	Reverse Small
Z	Zero
FS	Forward Small
FM	Forward Medium
R	Right
ST	Straight
L	Left

The input fuzzy sets are shown in figure 14.

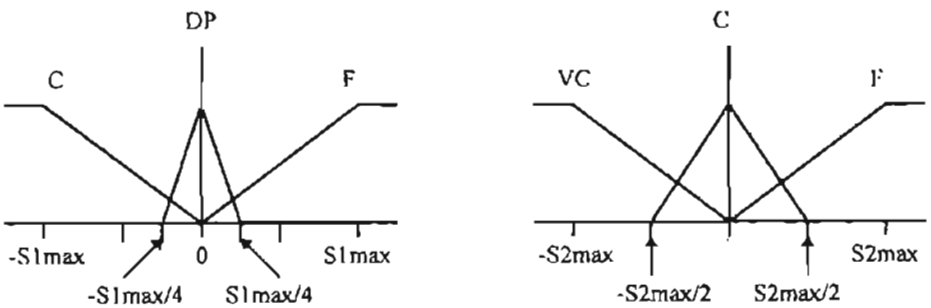


Figure 14. Input fuzzy sets for FLC (the ones for S3 are equal to the S2's)

The output fuzzy sets are defined in figure 15.

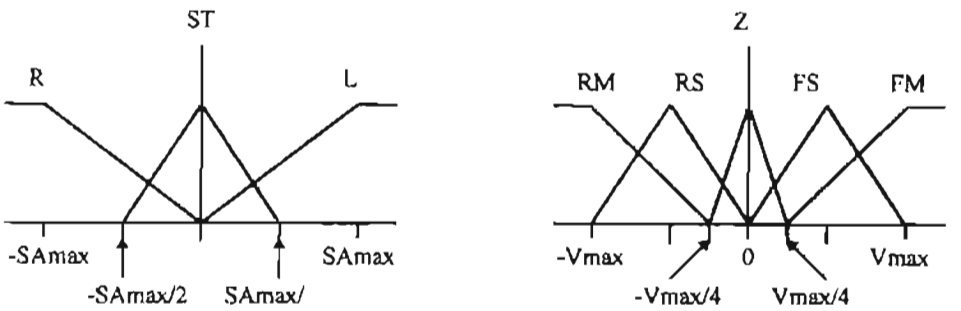


Figure 15. Output fuzzy sets for FLC

The output fuzzy sets do meet the general requirements and characteristics for computational efficiency and efficient use of memory. The shape of the membership functions is the triangle and is implemented by the following code:

```
/* output fuzzy sets and half of their support */

speedrm=-speedmax;
speedrs=-speedmax/2;
speedz=0;
speedfs=speedmax/2;
speedfm=speedmax;

sphsm=3*speedmax/4;
sphss=speedmax/2;
sphsz=speedmax/4;
```

The triangular shape of each fuzzy term is realized easily by a parametric, functional description of the membership function, that can be stored with minimal use of memory, manipulated efficiently in terms of real-time requirements, by the other modules of the controller.

Other characteristics of these fuzzy sets are:

- The membership functions are symmetric (left width is equal to right width) which is recommended by Driankov [4].

- In this case the crosspoint ratio is 1 and the crosspoint level is around 0.5 which are the usual choices in the literature [4].

- Also the condition width is met.

Rule Base

As needed for the experimentation, several rule bases were done always based on the knowledge and experience of the controller designer, which are expressed in the language of fuzzy if-then rules, for example:

if S1 is C, S2 is VC and S3 is VC then V is RS and SA is ST

In other words, each rule is defined according to the way the engineer believes the AV will follow the moving object from a desired distance, so if S1, S2 and S3 indicate the object is close then the AV will have to move slowly in reverse and in a straight way.

Since there are 3 linguistic values for each term set corresponding to the three inputs, so there are 3*3*3=27 rules.

Part of the code that implements the rule base is:

```
/* speed fuzzy numbers */
rsp[0][0][0]=speedrs; rsp[0][0][1]=speedrs; rsp[0][0][2]=speedrs;

/* speed fuzzy sets half supports */
rsps[0][0][0]=sphss; rsps[0][0][1]=sphss; rsps[0][0][2]=sphss;

/* steering angle fuzzy numbers */
rst[0][0][0]=stanst; rst[0][0][1]=stanst; rst[0][0][2]=stanl;

/* steering angle fuzzy sets half supports */
rsts[0][0][0]=stahsst; rsts[0][0][1]=stahsst; rsts[0][0][2]=stahsl;
```

The rule base is defined as two pairs of matrixes of 3*3 (labeled rsp-rsps and rst-rsts) which are easy to read, modify and access by means of indexes or pointers. One pair for V and the other for SA. For given S1, S2 and S3 there is one entry in each to the two matrixes of the pairs. For example for the position [0][0][0] of the matrixes (case of the rule shown above), where S1=C and S2=S3=VC, the entries in the matrix pair for V are "speedrs" and "sphss", and for SA "stanst" and "stahsst"; which correspond to the

actual output fuzzy numbers for the corresponding rule consequent (V is RS and SA is ST) and the half of the magnitudes of the supports for those output fuzzy sets ("sphss" and "stahsst"), which are, both, values used during the defuzzification.

Process Logic

The inference engine or rule firing can be of two types [4]:

a) composition based-inference or CBI(where a set of rules is fired via composition)

b) individual rule-based inference or IRBI(where each individual rule is fired separately).

The FLC of the Supervisor uses the type b.

IRBI can be described by the following 3 steps:

1) Computing the degree of match, also known as weighting factor or firing strength and labeled **fi**, between the crisp inputs and fuzzy sets describing the meaning of the rule antecedent. There are two methods to calculate **fi** according to Lee[12]: The first uses the minimum operation in the Cartesian Product, since it is widely used in FLC applications, it is chosen for the FLC. The second employs the algebraic product in the Cartesian Product.

2) Clipping the fuzzy set describing the meaning of the rule consequent to the degree to which the rule antecedent has been matched by the crisp inputs.

3) Finally the clipped values for the control output of each rule are aggregated, thus forming the value of the overall control output.

If IRBI is implemented by using:

a) the product operation rule of fuzzy implication (Larsen's rule Rp where Rp: a-> b=a.b) to represent each rule of the rule base

b) max-min composition performed between the fuzzified input and an individual rule, and

c) **fi** is the minimum operation;

then, giving the inputs S1, S2 and S3 as the singletons s1, s2 and s3 respectively and the output fuzzy sets μ_{LVl} and μ_{LSAl} of the nth rule's consequent, then the fuzzy control action is given by:

(7) $$\mu_{LV'} = \bigcup_{i=1}^{n} fi * \mu_{LVi}$$

(8) $$\mu_{LSA'} = \bigcup_{i=1}^{n} fi * \mu_{LSAi}$$

The proof of these results are given by Lee [12]

where: n=27 rules and

(9) $$fi = \min [\mu_{LS1}(s1), \mu_{LS2}(s2), \mu_{LS3}(s3)]$$

which is a measure of the contribution of the nth rule to the fuzzy

control action.

The rule firing module correspond to the following code:

```
/* Process Logic */
for (p=0;p<nx1;p++)
        for (q=0;q<nx2;q++)
                for (r=0;r<nx3;r++)
                        fi[p][q][r]=mini(mux1[p],mux2[q],mux3[r]);
```

where the nx1*nx2*nx3 = 27 firing strengths are saved in a matrix fi 3 by 3 by 3.

The non-fuzzy control outputs $\mu_{LV'}$ and $\mu_{LSA'}$,corresponding to the equations (7) and (8) are obtained and explained in the section of Defuzzification.

Fuzzification

According to Driankov [4] when individual rule based inference is used, then the fuzzification should be implemented as explained next. For example, if s2 is crisp input, then the fuzzified version s2* is its degree of membership in μ_{LS2} or $\mu_{LS2}(s2^*)$. In the code, this is implemented as:

```
mux2[1]=memfun(s2hsc,s2c,son2);
```

where son2 is s2, mux2() is $\mu_{LS2}(s2^*)$ and 'memfun' is a function that calculates the degree of membership for the triangular shape of a fuzzy set.

Defuzzification

The chosen defuzzification method is called Quality Method [10] or Modified-Center-Average-Defuzzifier [28]. As explained by Wang this is the more suitable method, because it is important to take into account the support of output fuzzy sets and also their heights that is, we need a method with both vertical and horizontal component. According to Hellendoorn [10] the main disadvantage of the other methods is that they ignore the fact that rules with crisper output are more important than those with fuzzier outputs. So the formula is the following:

$$(10) \qquad v^* = \frac{\sum_{i=1}^{n} fi * \mu_{LVi} / dVi}{\sum_{i=1}^{n} fi / dVi}$$

$$(11) \qquad sa^* = \frac{\sum_{i=1}^{n} fi * \mu_{LSAi} / dSAi}{\sum_{i=1}^{n} fi / dSAi}$$

where μ_{LV^*}, μ_{LSA^*}, μ_{LVi}, μ_{LSAi}, fi and n are the same defined in section of Process Logic; dVi and dSAi are the supports of the corresponding output fuzzy sets of the nth rule consequent. It is easier to understand these equations because of its similarity to equations (7) and (8).

This method is implemented in the FLC by the following code:

```
/* defzzification */
sumNsp=sumDsp=sumNan=sumDan=0;
for (p=0;p<nx1;p++)
        for (q=0;q<nx2;q++)
                for (r=0;r<nx3;r++)
                {
                        if (fi[p][q][r]) {
                        sumNsp = sumNsp + rsp[p][q][r]*fi[p][q][r]/rsps[p][q][r];
                        sumDsp = sumDsp + fi[p][q][r]/rsps[p][q][r];
```

```
                    sumNan = sumNan + rst[p][q][r]*fi[p][q][r]/rsts[p][q][r];
                    sumDan = sumDan + fi[p][q][r]/rsts[p][q][r];
                    }
                }

RoadSpeed = (int)(sumNsp/sumDsp);
....
RoadSpeed += 32767;
....
SteeringAngle = (int)(sumNan/sumDan);
```

As it is shown, for example in the case of the speed, both the output fuzzy set μ_{LVi} of the nth rule consequent and its support dVi are gotten by accessing the rule-base matrixes (in an easy way since both values are located in the same position in the matrixes). The firing strength is obtained from the matrix "**fi**".

In this code there is an improvement regarding efficiency (faster execution) because the operations in the loop occur only if the firing strength fi is different than zero. (This is an important feature to consider specially when it is known that only 8 rules may be, at most, fired each time, every sampling period).

This method of defuzzification is also preferred because it performs very well the following properties: continuity, disambiguity, plausibility, computational complexity, weight counting and quality regarding as defined by Hellendorn [10], and it is the only one of seven common methods that performs all six properties according to the same author.

CHAPTER IX

AV'S PERFORMANCE

Integration of AV's Modules

The integration of the five modules of the AV and the definition and implementation of interactions among the modules is a process of gradual progress and evolution, but at the same time a process of continuous reformulating of hypothesis for the design of the subsystem. This reformulating process, which is actually the redesign or tuning of each AV's subsystem is more necessary after the design

TABLE 8a.

AV'S HARDWARE Part I [19, and additional resources]

Hardware Description	Quantity	Specifications
Everest & Jennings Child's Wheelchair - Hot Wheels Line SR- M-109233	1	Power Supply - 24 Volt DC Motors - 24 Volt Permanent Magnet with electromagnetic brakes Purchased from: Loyal LaPlante Supply Co. 6702 East 11th Street Tulsa, OK 74112
Everest & Jennings Battery Charger 85000	1	24 Volt Solid State Self Regulating Charger
Beckman DC Tachometer	2	Output: 6.5 Volts/1000 RPM
BEI Rotary Optical Encoder	2	Resolution: 120 Counts/Revolution 5 Volt Supply, 60 ma. max. 200 RPM max. shaft speed
Polaroid Ultrasonic Ranging Unit (Sonar Units)	6	5 Volt Supply, 40 ma Range: 40cm - 500cm Error : +/-2cm Beam Width: 15 Degrees

of the modules as independent units and it becomes even more necessary as soon as the interactions among two or more modules begin to be analyzed, reaching its maximum degree of necessity during the experimentation of all modules integrated as a whole, in other words the AV itself.

TABLE 8b.

AV'S HARDWARE Part II

Description	Quaintly	Comment
ENAT board	3	Used as the Propulsion, Vision and Gateway microcontrollers
Emergency Stop Button	1	Electrically disconnects the wheelchair's electronics from the DC motors when depressed.
Added Components and Electronics for desired control	-	2 Digital to Analog Converters, 4 Operational Amplifiers, 4 Low Pass Filters, 2 Mechanical Relays
Registers 74LS377	2	Used to implement the Demultiplexing Section for the Propulsion Module

In figure 16 there is a block diagram of the AV illustrating the integration of its hardware and software and in tables 8 and 9 there appear the AV's hardware and software specifically.

Figure 17 shows the AV as it looks by the end of this research.

TABLE 9

AV'S SOFTWARE

Software Name	Application
HC11 Assembly Language (by Motorola)	CAN routines
Small C (by James Hendrix, with new functions created by New Micros, Inc.)	3 programs for the 3 ENAT boards: drive4.c, sonar1.c and gate1a.c
Turbo C (by Borland)	1 program for the PC: sup4.c (And Andujar and Shanley's modified code)
CFLOW (by Ellington and Steeger)	Organization of complex C programs (Andujar and Shanley's code)
PROCOMM (by Datastorm Technologies, Inc.)	Terminal Emulator for HC11
BUFFALO (by Motorola)	Monitor program for HC11

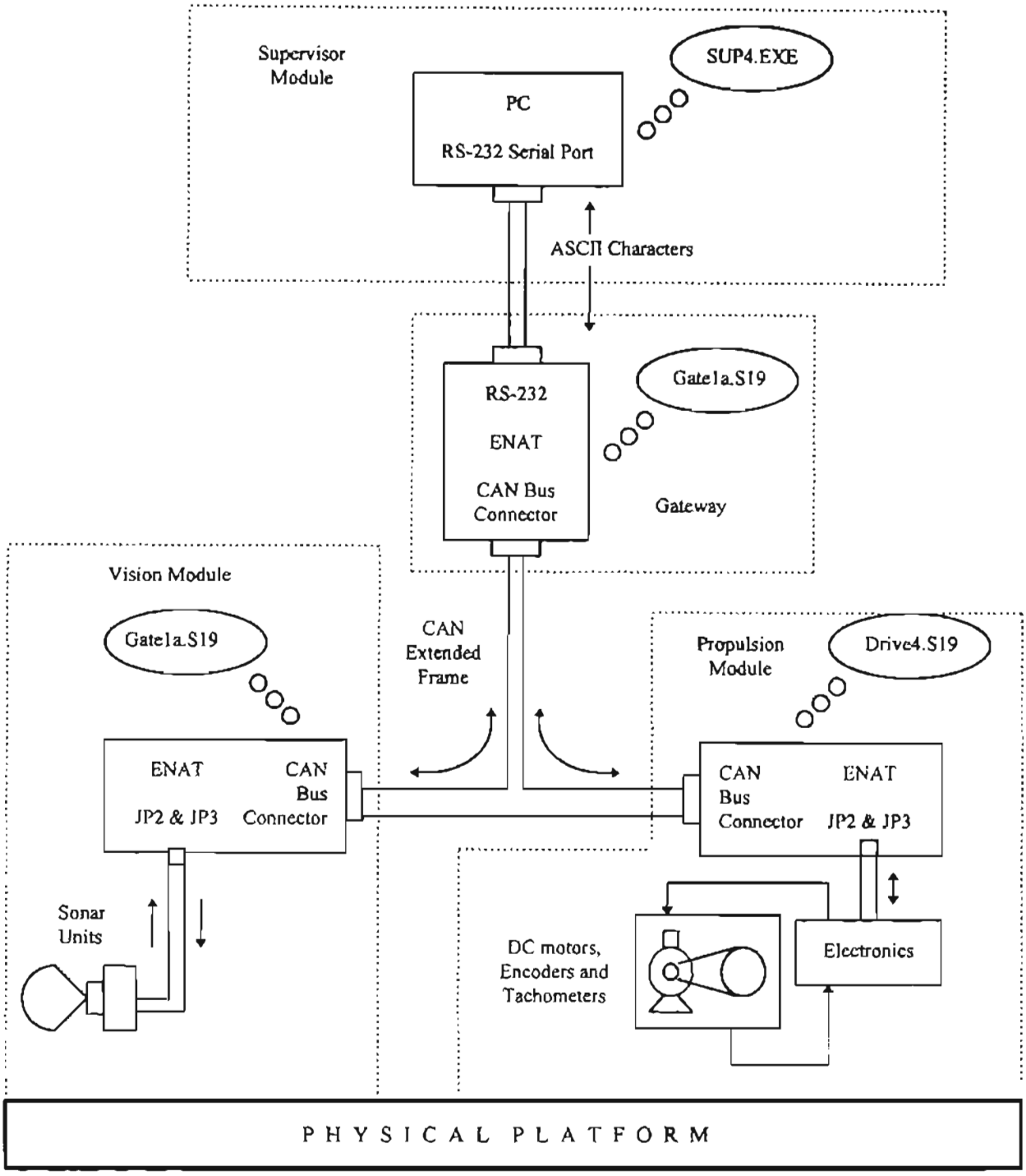


Figure 16. AV's Integration of Hardware and Software

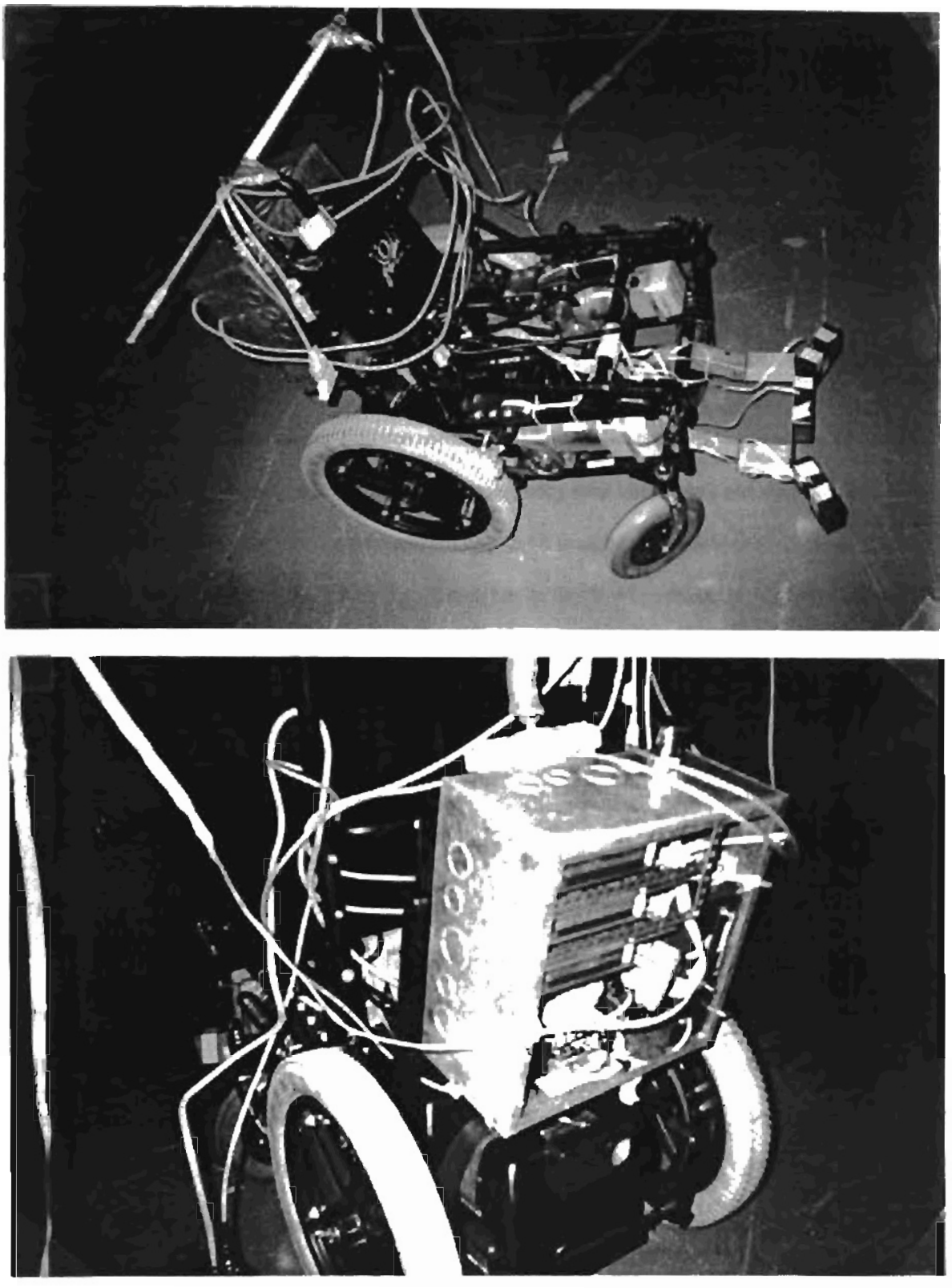


Figure 17 AV (top)
AV's detail: Computer
Box (bottom)

AV Operating Procedure

Part of this procedure includes Newton's operating procedure [19]. Even though the maximum speed of the wheelchair has been set at 3 feet per second (approximately 40 rpm), it is still very powerful. So the operation of the AV must be done adhering to the following guidelines when powering up the wheelchair and operating it.

1) Manually disengage the DC motors from the drive wheels by lowering the levers just inside each wheel. This action will ease the tension in each drive belt.

2) Make sure the emergency stop connection is made from the computer box to the wheelchair electronic box. **If this connection is not made, the emergency stop button will not stop the unit.** It is labeled as the "emergency stop connection" and at this point it must be depressed.

3) Place the "high/low" switch on the joystick box in the "low" position. If this switch is set in the "high" position, the computer may not provide a stable response.

4) Place the "computer/joystick" switch in the "computer" position. This switch has two other positions, one for the joystick and one for neutral.

5) Place the "on/off" switch in the "on" position. This action will activate all the electronics.

6) Make sure the switches in the ENAT boards are as follows. SW1: WENABLE position. SW2: PC mode. SW4: Power ON.

7) The ENAT boards mounted in the computer box are labeled Propulsion, Vision and Gateway and their positions are bottom, middle and top respectively. (Refer to the M68HC11EVBU Universal Evaluation Board Users Manual, page 4-39, for instructions on downloading the computer code). The executable files for the Propulsion, Vision and Gateway controllers are "DRIVE4.S19", "SONAR.S19" and "GATE1A.S19", respectively. The starting memory address of the three programs is "20F0".

8) With PROCOMM running in the PC proceed to download the Propulsion and Vision codes. Once the download is complete type "g 20F0" to run each program, the order here does not matter. At this point the sonar units should be making a clicking sound.

9) Download the Gateway code ("GATE1A.S19") and leave the cable of the SCS to establish serial communications with the Supervisor (PC). Type "g 20F0" to run the program.

10) Exit PROCOMM and run the Supervisor executable file, SUP4.EXE.

11) Manually engage the belt drives.

12) Place the moving object at desired position

13) Release the emergency stop button (lift it up).

14) Begin to move the moving object slowly and when needed press the emergency stop button to stop the AV.

Experimentation

Table 10 describes the experiments done to analyze the performance of the AV. For all of them the AV had to follow a moving object describing an "S" shaped trajectory. After the table graphs are shown to illustrate some of the experiments that are representative of the AV's performance.

TABLE 10A

EXPERIMENTS WITH DIFFERENT "fi" PRODUCTION (all distances in cm)

Exp. #	D. Dis.1	D. Dis. 2	D. Dis. 3	S1 max.	S2 max.	S3 max.	"fi" production
1	90	160	160	40	90	90	Minimum operation
2	90	160	160	40	90	90	Algebraic product

TABLE 10B

EXPERIMENTS VARYING DESIRED DISTANCES AND MAXIMUM VALUES (all distances in cm)

Exp. #	D. Dis.1	D. Dis.2	D. Dis.3	S1 max.	S2 max.	S3 max.	V max.	SA max.
3	90	90	90	60	60	60	30	45
4	90	150	150	40	90	90	30	45
5	90	160	160	40	90	90	30	45
6	150	100	100	90	40	40	30	45
7	90	160	160	40	90	90	15	45
8	90	160	160	40	90	90	40	45
9	90	160	160	40	90	90	30	60
10	90	160	160	40	90	90	30	30

TABLE 10C

EXPERIMENTS VARYING FUZZY SETS SUPPORTS
(Experiments 1 to 10 use the following halves of supports:
S1: DP: S1max/4, C: S1max/2. S2-S3: C: S#max/2.
V: Z: Vmax/4, XM: 3*Vmax/4. SA: ST: Samax/2.)

Exp. #	Halves of Supports Definition (all other parameters as in Exp. 5)
11	Same supports for inputs as Exp. 1 to 10 and V: Z: Vmax/5, XM: 4*Vmax/5
12	As Exp. 11 and V: Z: Vmax/2, XM: Vmax/2
13	As Exp. 11 and SA: ST = L= R: SAmax
13a	All fuzzy sets with constant supports (Height Method for Defuzzification)
14	As Exp. 11 and V: Z: Vmax/4, XM: 3*Vmax/4; SA: ST: SAmax

TABLE 10D

EXPERIMENTS VARYING RULES
(the modified rules are indicated according to the corresponding matrix's entry of "SUP4.C" code)

Exp. #	Rules Definition (all other parameters as in Exp. 5)
15	rsp: [0][0][0]=[0][2][0]= [0][0][2]=RM, [2][2][2]=FM rst: [0][2][0]= [0][2][1]= [0][0][2]= [0][1][2]=ST
16	rsp: [0][0][0]=[0][2][0]= [0][0][2]=RS
17	rst: [0][2][0]= [0][2][1]= R, [0][0][2]= [0][1][2]=L
18	rsp: [2][2][2]=FS

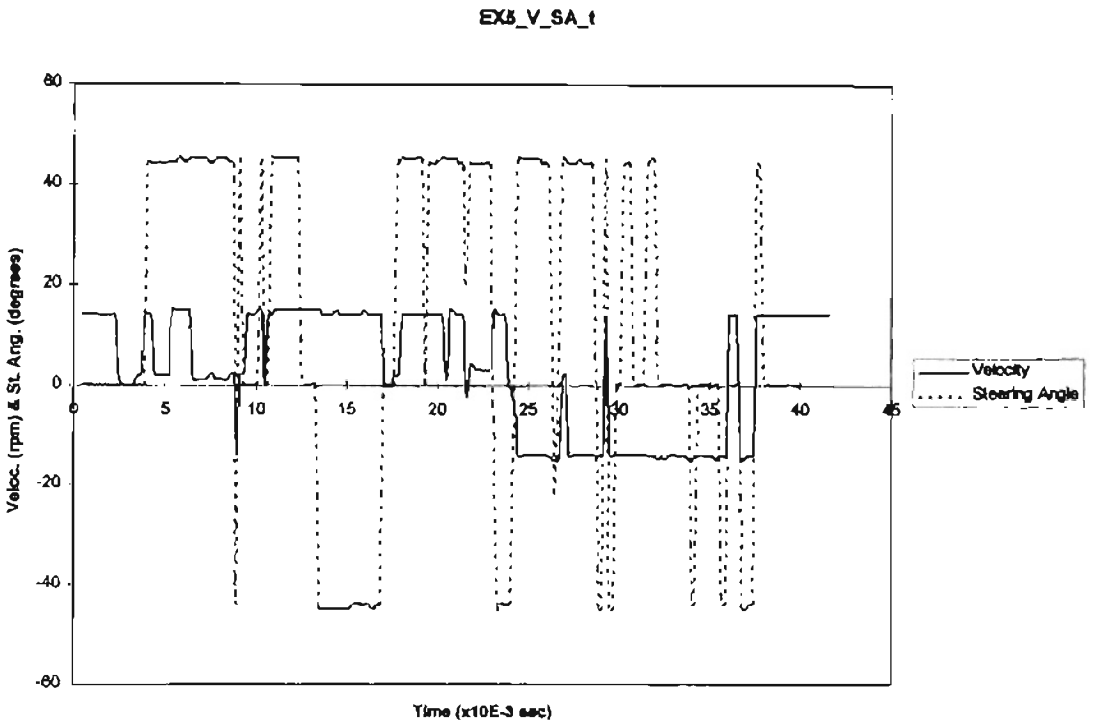


Figure 18 Velocity and steering angle vs. time according to Exp. 5

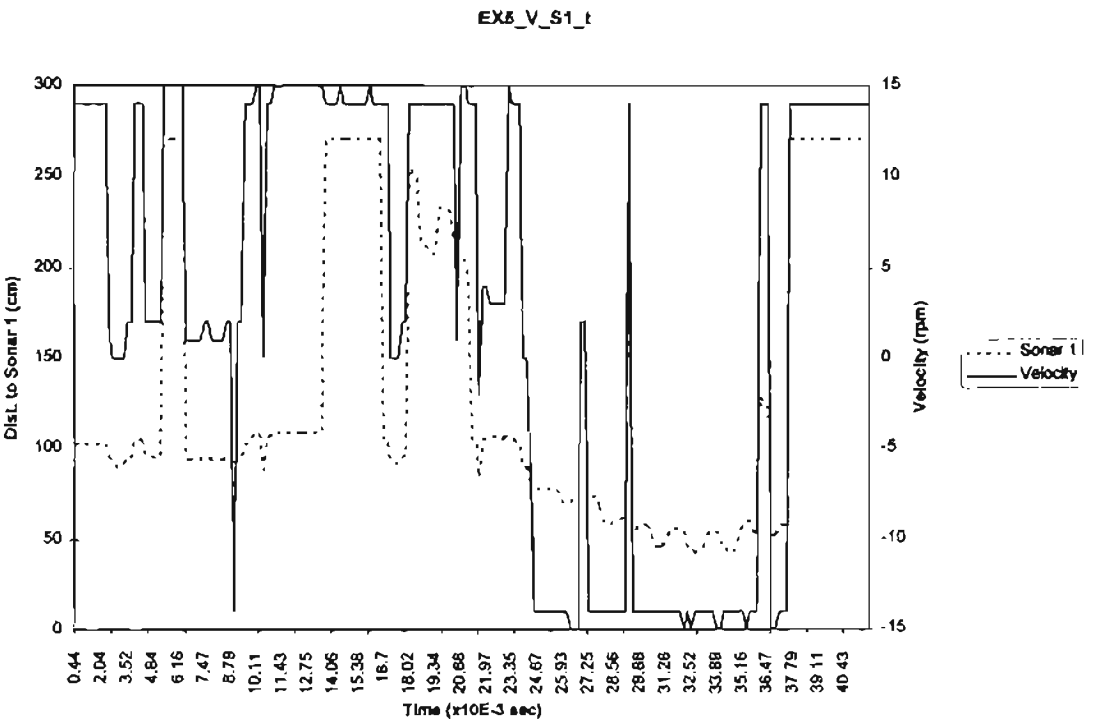


Figure 19. Distance between moving object and sonar 1 and velocity vs. time according to Exp. 5

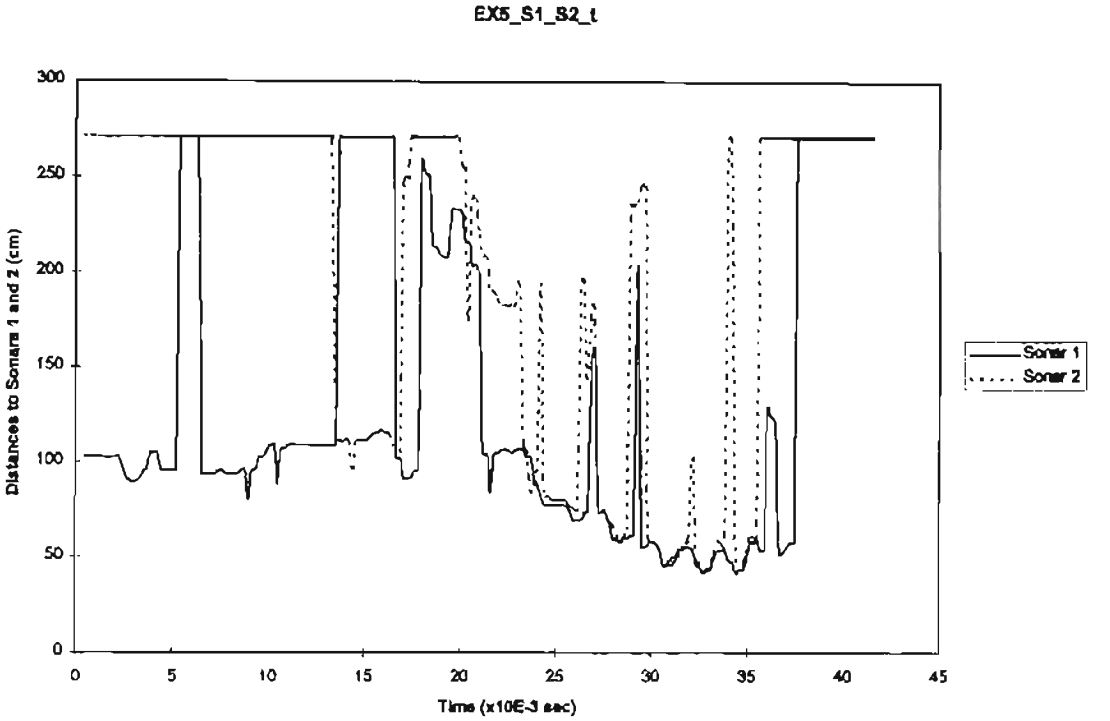


Figure 20. Distances between moving object and sonars 1 and 2 vs. time according to Exp. 5

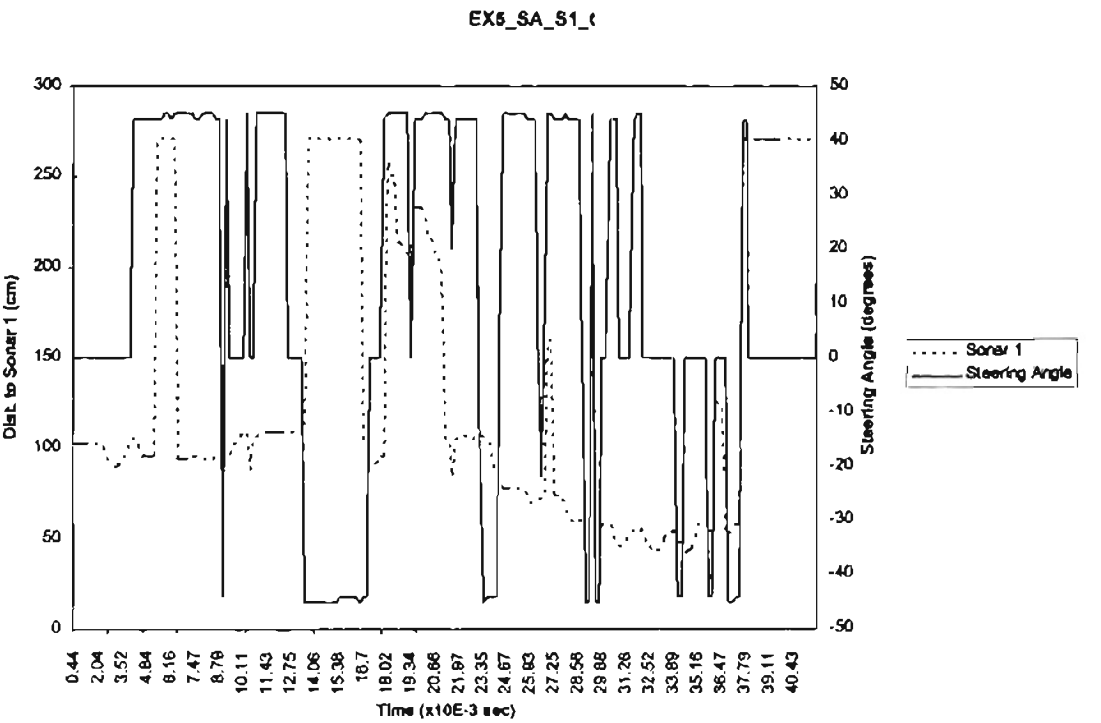


Figure 21. Distance between moving object and sonar 1 and steering angle vs. time according to Exp. 5

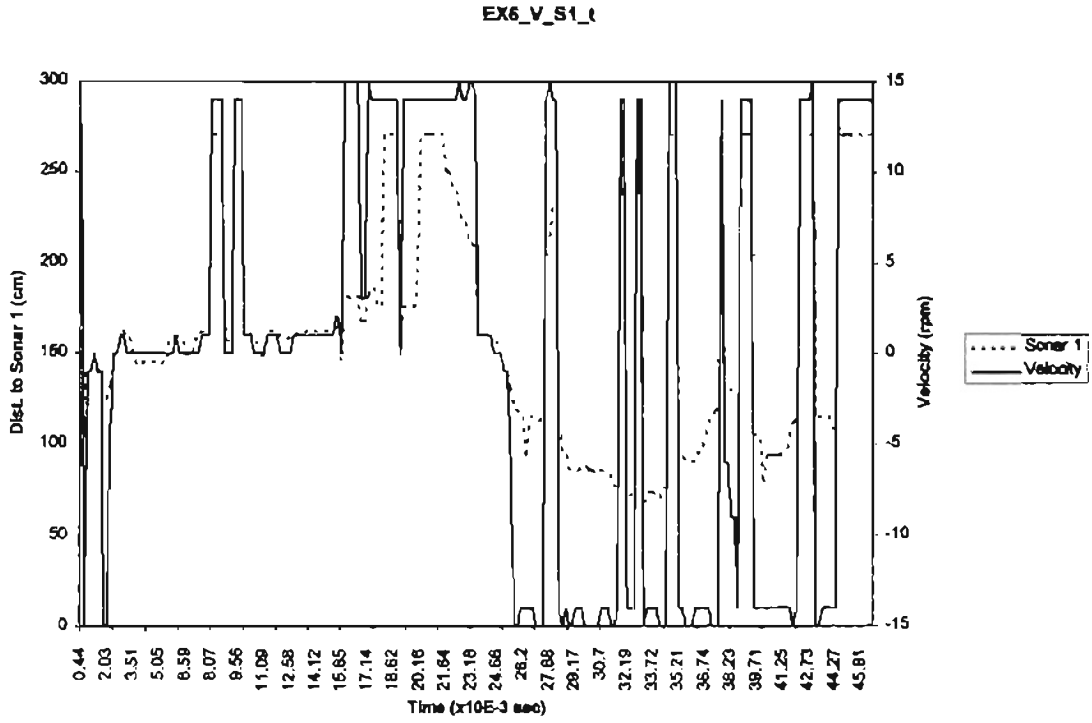


Figure 22. Distance between moving object and sonar 1 and velocity vs. time according to Exp. 6

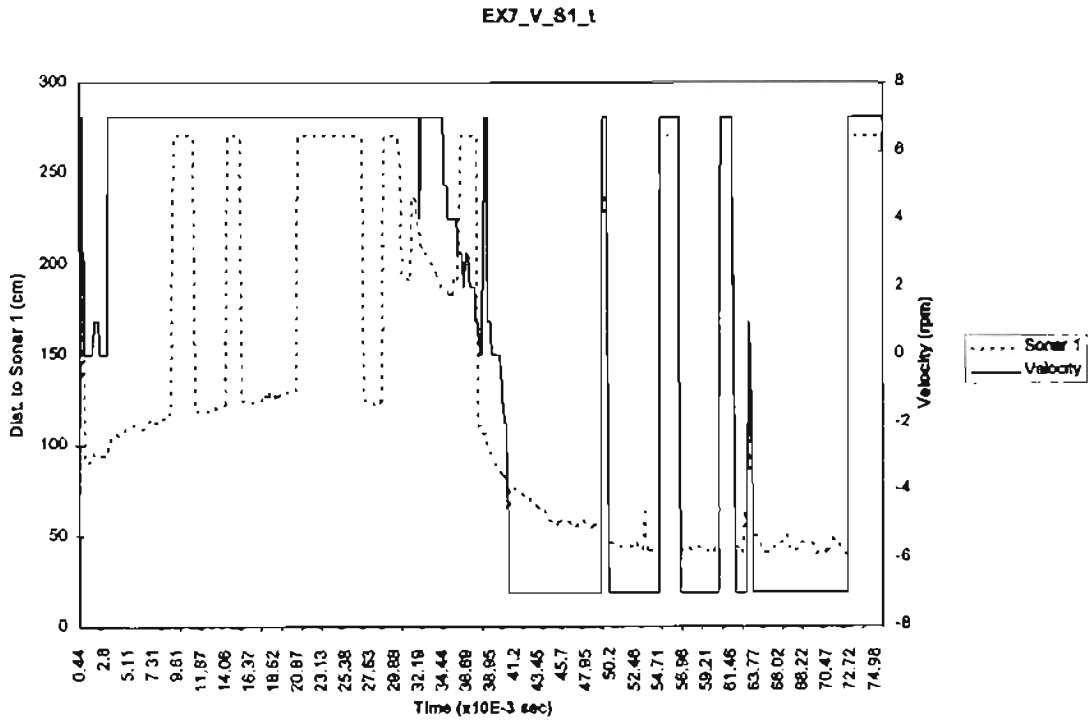


Figure 23. Distance between moving object and sonar 1 and velocity vs. time according to Exp. 7

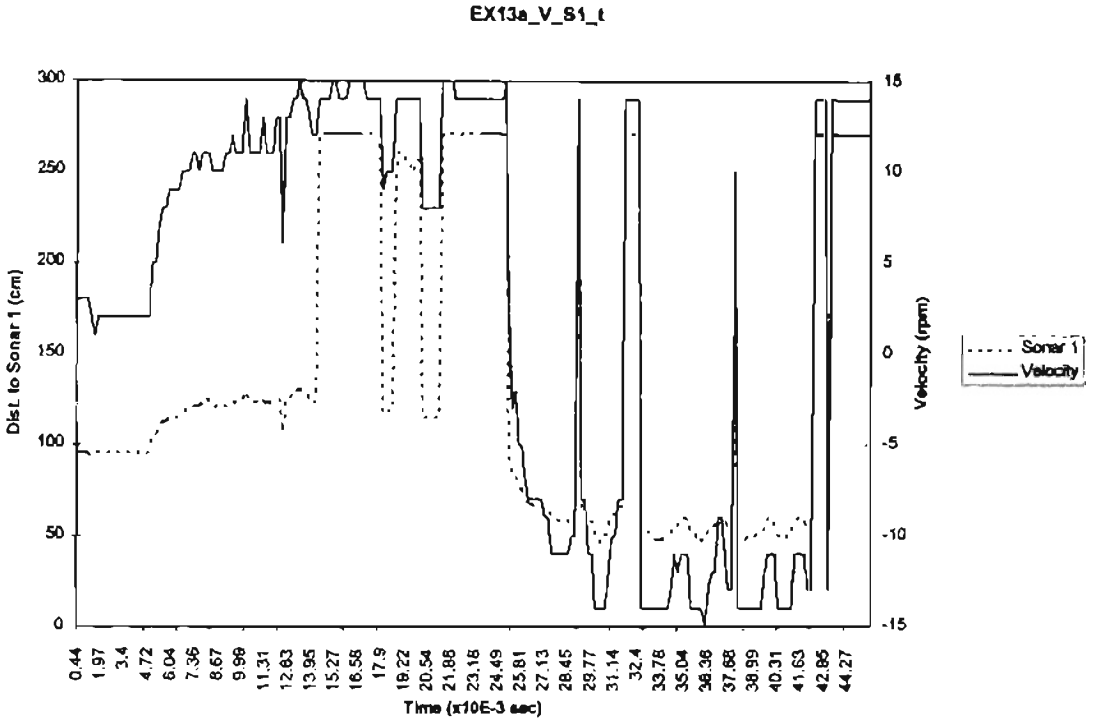


Figure 24 Distance between moving object and sonar 1 and velocity vs. time according to Exp. 13a

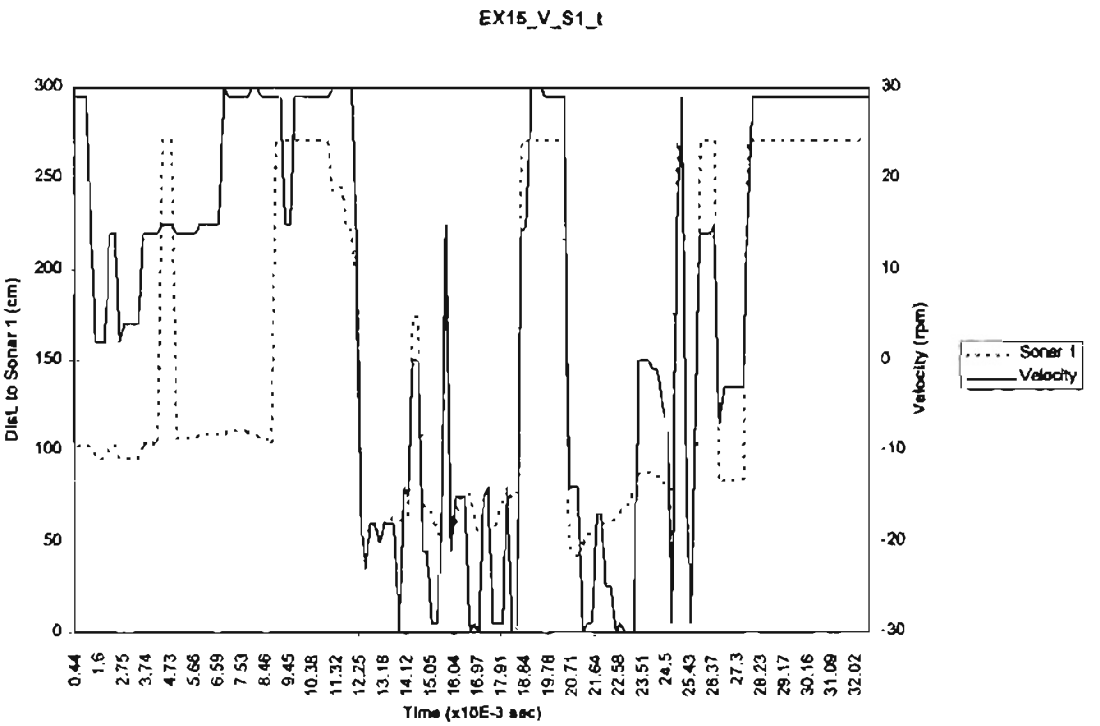


Figure 25. Distance between moving object and sonar 1 and velocity vs. time according to Exp. 15

Discussion of Results

The performances of the AV using two different ways of calculating the firing strength fi were the same in both cases (experiments #1 and #2, table 10A).

In order to tune up the FLC there are several parameters that can be adjusted. Among these there are the scaling factors with which controller input and output values are mapped onto the universe of discourse of the fuzzy set definition. This is equivalent to modifying the maximum values corresponding to the limit values of the universe of discourse: S1 max, S2 max, S3 max, V max and SA max.

Altering the maximum values changes the classification of the input value. For example, for maximum value of Vmax = 30 a speed of 15 rpm is classified as FS, but for maximum value of Vmax = 15 it is classified as FM.

The experiments #3 to #10 (table 10B) correspond to variations in maximum values for input and output fuzzy sets. In #3 the AV looses the moving object very fast, because it reacts too fast and violently going backwards and this makes the AV to move the sonars in such a way that they cannot "see" the moving object. #4 performs much better than #3. #6 (figure 22) is a very hard experiment to run since the desired distance is pretty big and the AV will see other objects in the lab and very quickly will try to get them (as if they were the moving object); besides even if the moving object is not very close and moving toward the AV it moves backwards very fast and dangerously. #7 (figure 23) shows a very slow movement going forward, but it is satisfactory; it shows the best AV's performance going backwards. #8 offers the best performance going forward (but looses the moving object going backwards). #9 and #10 demonstrate that the change in maximum value of steering angle affects very little the performance. #5 is the best performance of all (figures 20 and 21) since the tasks are executed more precisely and effectively.

The experiments #11 to #14 (table 10C) correspond to the variation on supports. #11, #12, #13 and #14 show satisfactory performances in general. #13a (figure 24) uses all supports constant (in other words, it implements the Height Method of Defuzzification), and going forward the quality of performance decreases, turning to the left or right is very slow and inaccurate, although going backwards is fine.

The experiments #15 to #18 (table 10D) correspond to rule changes in the rule base matrixes. #15 (figure 25) has rules that use RM and FM in consequents; so going backwards or forward the speed is too high, although this makes the AV able to follow fast moving objects (specially going forward). #16 replaces in three rules RM by RS and the performance is only improved going backwards. #17 changes in four rules the corresponding steering angle and shows a better performance than #16. #18 incorporates all the above and changes only one rule (FM by FS).

In all the experiments above there appeared a conflict between the two tasks assigned to the AV: when the moving object passes near an obstacle, the AV is not able to tell which is the object and which the obstacle and while the moving object may continue moving away from the AV, this will stay in front of the obstacle. However the AV will indeed avoid the collision with the obstacle. Actually the AV will avoid collisions with obstacles, but only if they do not enter or appeared in the blind spot of the AV's sonar units (0 to 40cm around the wheelchair).

AV Operating Procedure

Part of this procedure includes Newton's operating procedure [19]. Even though the maximum speed of the wheelchair has been set at 3 feet per second (approximately 40 rpm), it is still very powerful. So the operation of the AV must be done adhering to the following guidelines when powering up the wheelchair and operating it.

1) Manually disengage the DC motors from the drive wheels by lowering the levers just inside each wheel. This action will ease the tension in each drive belt.

2) Make sure the emergency stop connection is made from the computer box to the wheelchair electronic box. **If this connection is not made, the emergency stop button will not stop the unit.** It is labeled as the "emergency stop connection" and at this point it must be depressed.

3) Place the "high/low" switch on the joystick box in the "low" position. If this switch is set in the "high" position, the computer may not provide a stable response.

4) Place the "computer/joystick" switch in the "computer" position. This switch has two other positions, one for the joystick and one for neutral.

5) Place the "on/off" switch in the "on" position. This action will activate all the electronics.

6) Make sure the switches in the ENAT boards are as follows. SW1: WENABLE position. SW2: PC mode. SW4: Power ON.

7) The ENAT boards mounted in the computer box are labeled Propulsion, Vision and Gateway and their positions are bottom, middle and top respectively. (Refer to the M68HC11EVBU Universal Evaluation Board Users Manual, page 4-39, for instructions on downloading the computer code). The executable files for the Propulsion, Vision and Gateway controllers are "DRIVE4.S19", "SONAR.S19" and "GATE1A.S19", respectively. The starting memory address of the three programs is "20F0".

8) With PROCOMM running in the PC proceed to download the Propulsion and Vision codes. Once the download is complete type "g 20F0" to run each program, the order here does not matter. At this point the sonar units should be making a clicking sound.

9) Download the Gateway code ("GATE1A.S19") and leave the cable of the SCS to establish serial communications with the Supervisor (PC). Type "g 20F0" to run the program.

10) Exit PROCOMM and run the Supervisor executable file, SUP4.EXE.

11) Manually engage the belt drives.

12) Place the moving object at desired position

13) Release the emergency stop button (lift it up).

14) Begin to move the moving object slowly and when needed press the emergency stop button to stop the AV.

Experimentation

Table 10 describes the experiments done to analyze the performance of the AV. For all of them the AV had to follow a moving object describing an "S" shaped trajectory. After the table graphs are shown to illustrate some of the experiments that are representative of the AV's performance.

TABLE 10A

EXPERIMENTS WITH DIFFERENT "fi" PRODUCTION (all distances in cm)

Exp. #	D. Dis.1	D. Dis. 2	D. Dis. 3	S1 max.	S2 max.	S3 max.	"fi" production
1	90	160	160	40	90	90	Minimum operation
2	90	160	160	40	90	90	Algebraic product

TABLE 10B

EXPERIMENTS VARYING DESIRED DISTANCES AND MAXIMUM VALUES (all distances in cm)

Exp. #	D. Dis.1	D. Dis.2	D. Dis.3	S1 max.	S2 max.	S3 max.	V max.	SA max.
3	90	90	90	60	60	60	30	45
4	90	150	150	40	90	90	30	45
5	90	160	160	40	90	90	30	45
6	150	100	100	90	40	40	30	45
7	90	160	160	40	90	90	15	45
8	90	160	160	40	90	90	40	45
9	90	160	160	40	90	90	30	60
10	90	160	160	40	90	90	30	30

TABLE 10C

EXPERIMENTS VARYING FUZZY SETS SUPPORTS
(Experiments 1 to 10 use the following halves of supports:
S1: DP: S1max/4, C: S1max/2. S2-S3: C: S#max/2.
V: Z: Vmax/4, XM: 3*Vmax/4. SA: ST: Samax/2.)

Exp. #	Halves of Supports Definition (all other parameters as in Exp. 5)
11	Same supports for inputs as Exp. 1 to 10 and V: Z: Vmax/5, XM: 4*Vmax/5
12	As Exp. 11 and V: Z: Vmax/2, XM: Vmax/2
13	As Exp. 11 and SA: ST = L= R: SAmax
13a	All fuzzy sets with constant supports (Height Method for Defuzzification)
14	As Exp. 11 and V: Z: Vmax/4, XM: 3*Vmax/4; SA: ST: SAmax

TABLE 10D

EXPERIMENTS VARYING RULES
(the modified rules are indicated according to the corresponding matrix's entry of "SUP4.C" code)

Exp. #	Rules Definition (all other parameters as in Exp. 5)
15	rsp: [0][0][0]=[0][2][0]= [0][0][2]=RM, [2][2][2]=FM rst: [0][2][0]= [0][2][1]= [0][0][2]= [0][1][2]=ST
16	rsp: [0][0][0]=[0][2][0]= [0][0][2]=RS
17	rst: [0][2][0]= [0][2][1]= R, [0][0][2]= [0][1][2]=L
18	rsp: [2][2][2]=FS

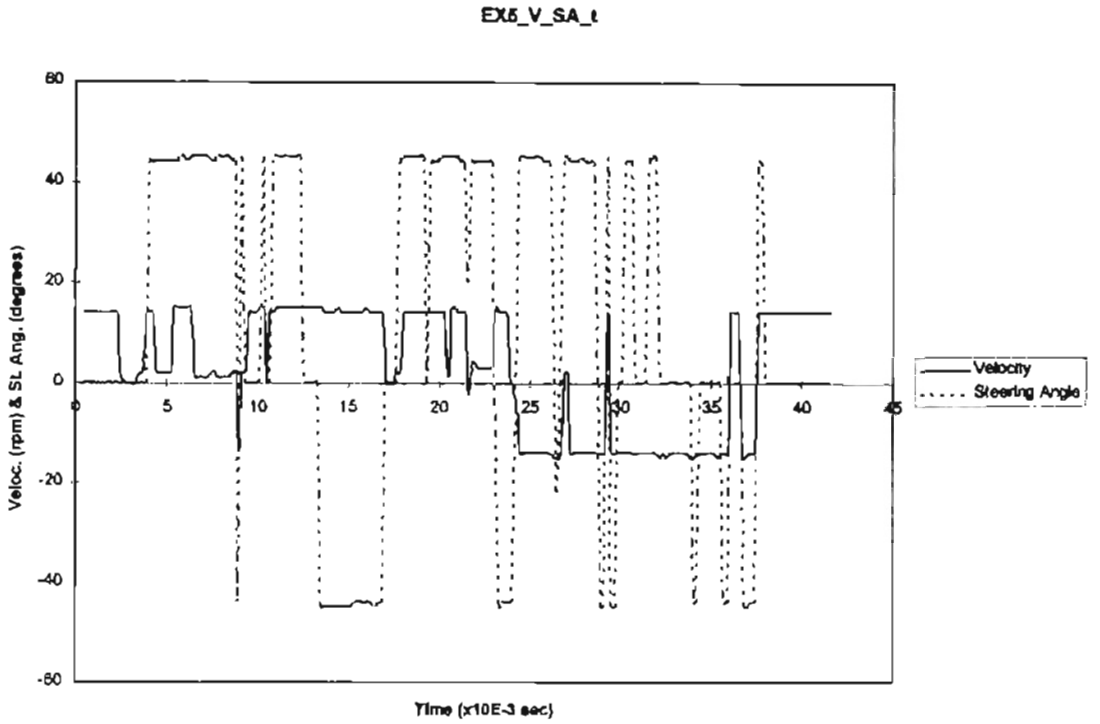


Figure 18 Velocity and steering angle vs. time according to Exp. 5

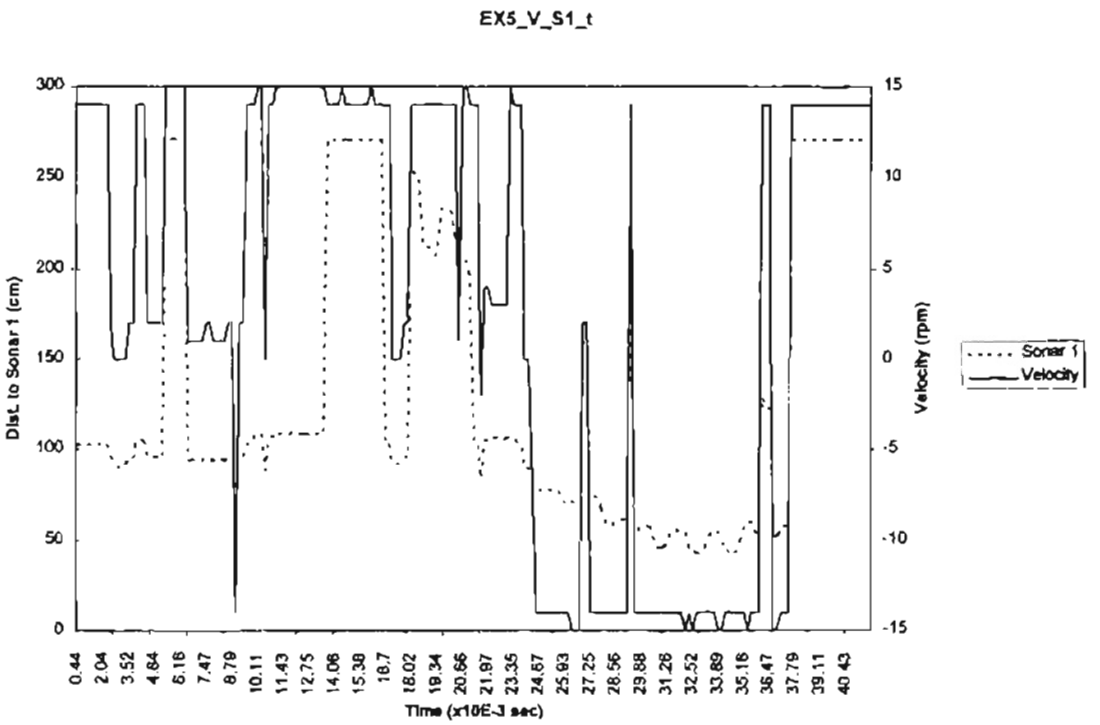


Figure 19. Distance between moving object and sonar 1 and velocity vs. time according to Exp. 5

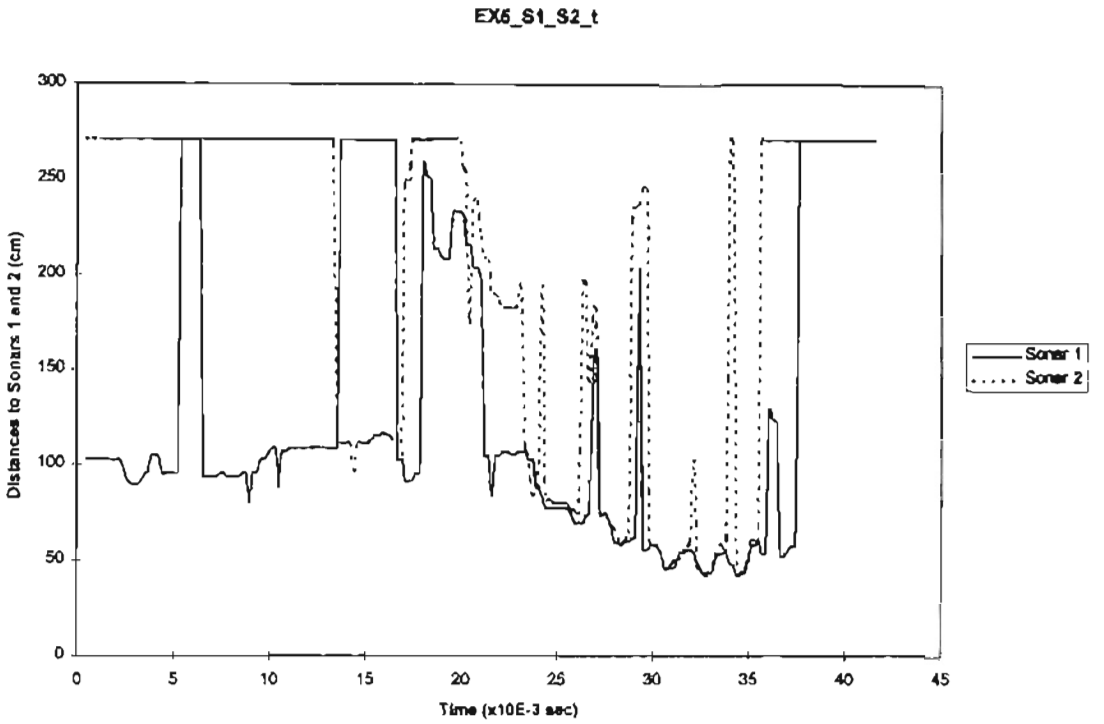


Figure 20. Distances between moving object and sonars 1 and 2 vs. time according to Exp. 5

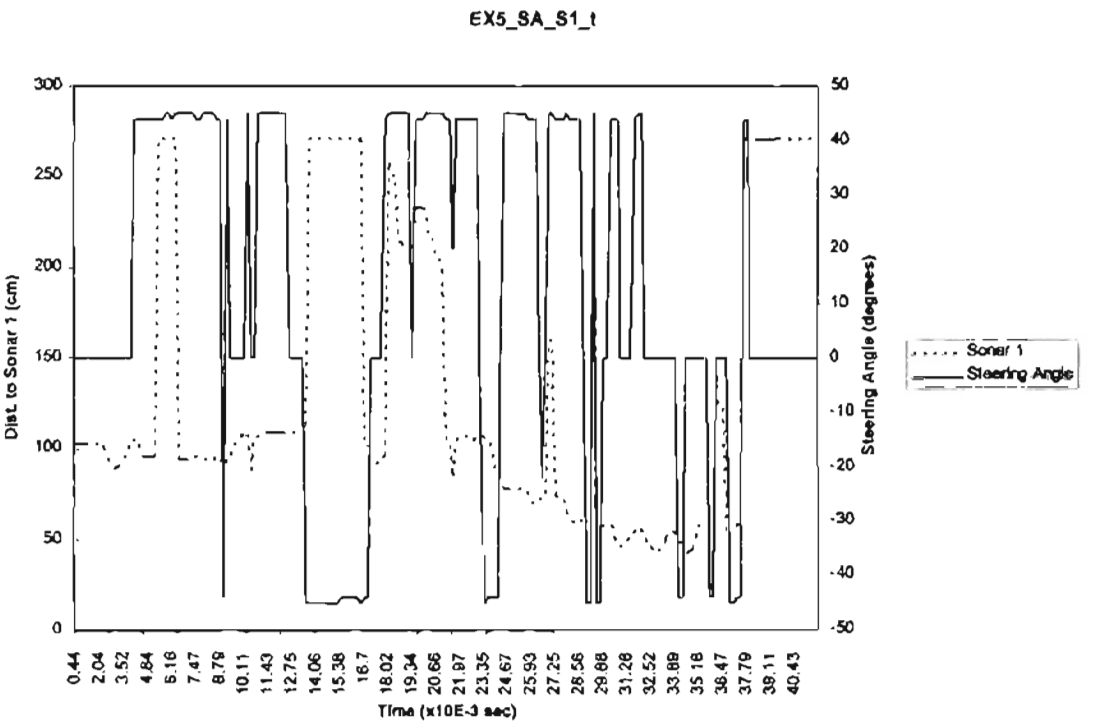


Figure 21. Distance between moving object and sonar 1 and steering angle vs. time according to Exp. 5

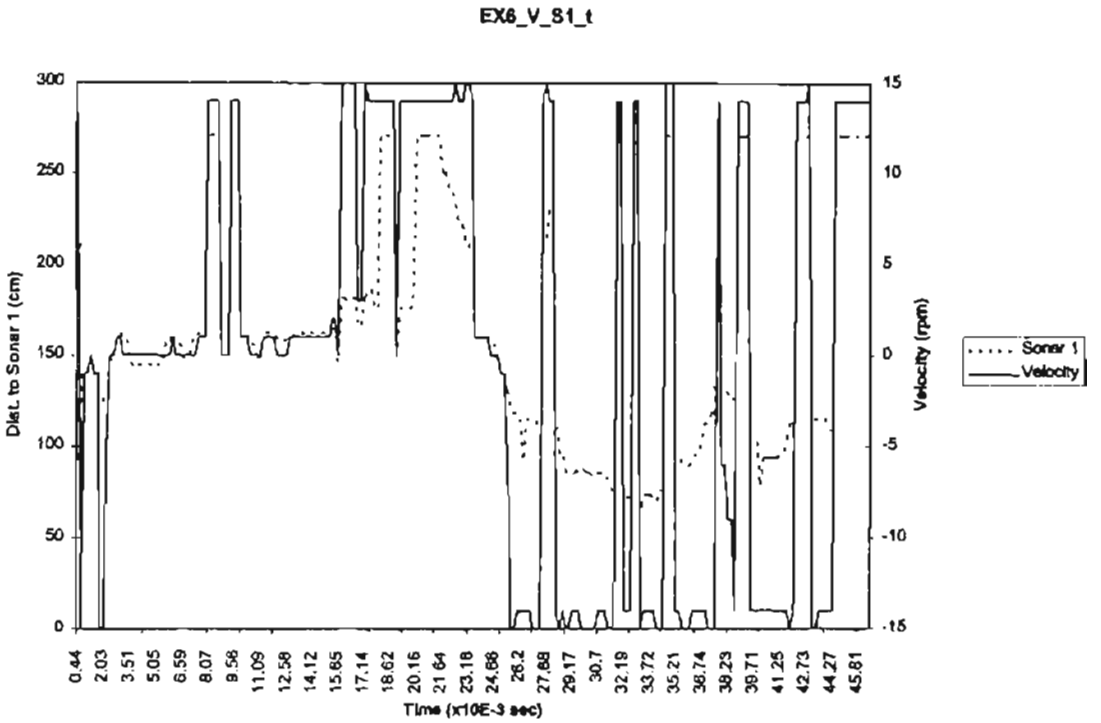


Figure 22. Distance between moving object and sonar 1 and velocity vs. time according to Exp. 6

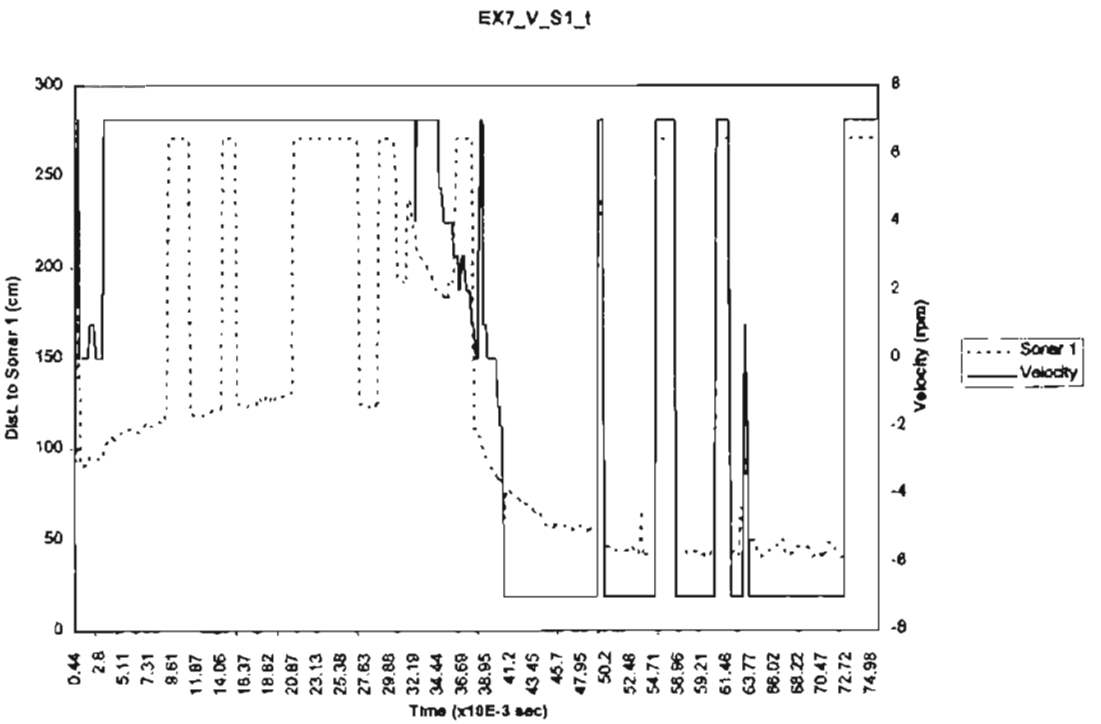


Figure 23. Distance between moving object and sonar 1 and velocity vs. time according to Exp. 7

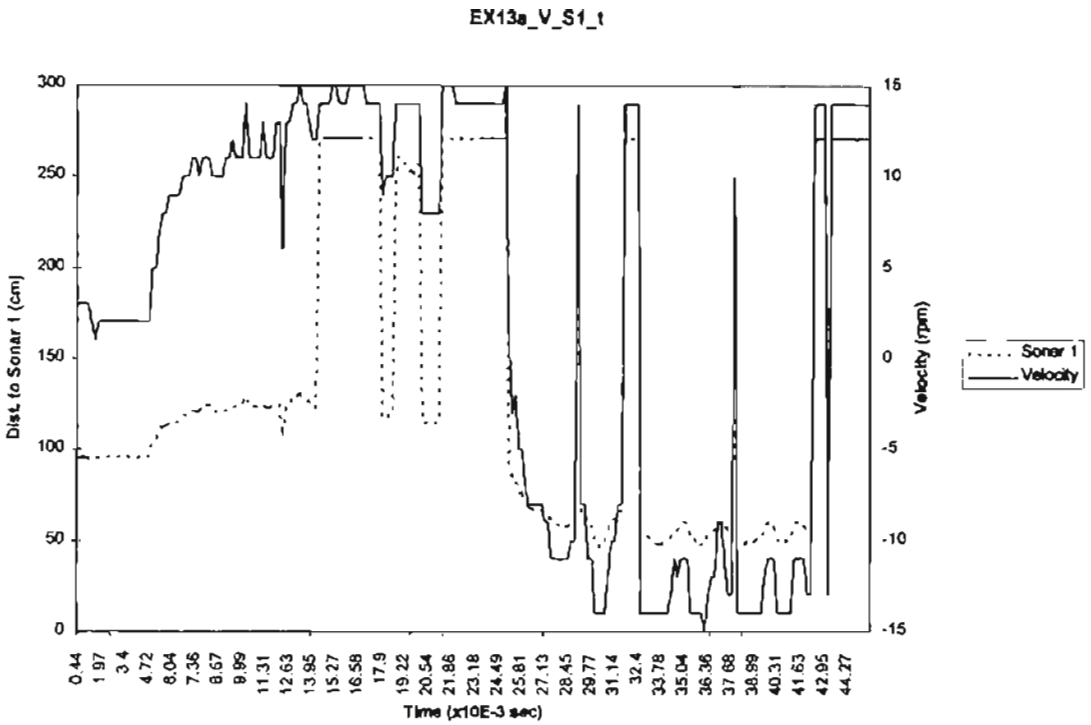


Figure 24 Distance between moving object and sonar 1 and velocity vs. time according to Exp. 13a

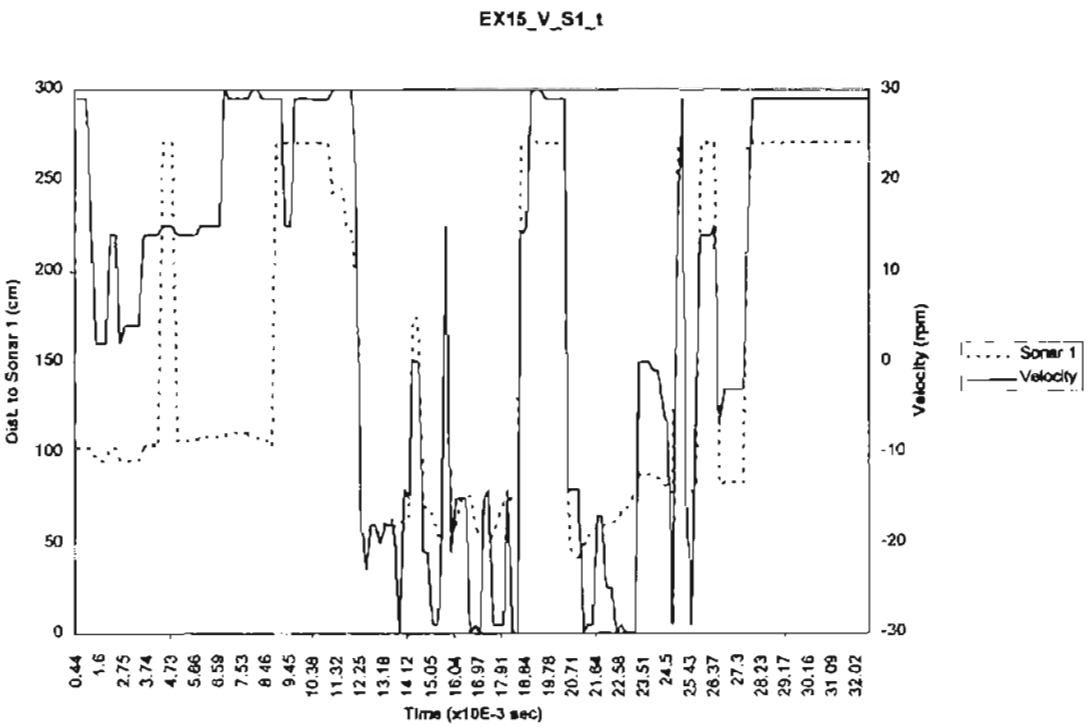


Figure 25. Distance between moving object and sonar 1 and velocity vs. time according to Exp. 15

Discussion of Results

The performances of the AV using two different ways of calculating the firing strength fi were the same in both cases (experiments #1 and #2, table 10A).

In order to tune up the FLC there are several parameters that can be adjusted. Among these there are the scaling factors with which controller input and output values are mapped onto the universe of discourse of the fuzzy set definition. This is equivalent to modifying the maximum values corresponding to the limit values of the universe of discourse: S1 max, S2 max, S3 max, V max and SA max.

Altering the maximum values changes the classification of the input value. For example, for maximum value of Vmax = 30 a speed of 15 rpm is classified as FS, but for maximum value of Vmax = 15 it is classified as FM.

The experiments #3 to #10 (table 10B) correspond to variations in maximum values for input and output fuzzy sets. In #3 the AV looses the moving object very fast, because it reacts too fast and violently going backwards and this makes the AV to move the sonars in such a way that they cannot "see" the moving object. #4 performs much better than #3. #6 (figure 22) is a very hard experiment to run since the desired distance is pretty big and the AV will see other objects in the lab and very quickly will try to get them (as if they were the moving object); besides even if the moving object is not very close and moving toward the AV it moves backwards very fast and dangerously. #7 (figure 23) shows a very slow movement going forward, but it is satisfactory; it shows the best AV's performance going backwards. #8 offers the best performance going forward (but looses the moving object going backwards). #9 and #10 demonstrate that the change in maximum value of steering angle affects very little the performance. #5 is the best performance of all (figures 20 and 21) since the tasks are executed more precisely and effectively.

The experiments #11 to #14 (table 10C) correspond to the variation on supports. #11, #12, #13 and #14 show satisfactory performances in general. #13a (figure 24) uses all supports constant (in other words, it implements the Height Method of Defuzzification), and going forward the quality of performance decreases, turning to the left or right is very slow and inaccurate, although going backwards is fine.

The experiments #15 to #18 (table 10D) correspond to rule changes in the rule base matrixes. #15 (figure 25) has rules that use RM and FM in consequents; so going backwards or forward the speed is too high, although this makes the AV able to follow fast moving objects (specially going forward). #16 replaces in three rules RM by RS and the performance is only improved going backwards. #17 changes in four rules the corresponding steering angle and shows a better performance than #16. #18 incorporates all the above and changes only one rule (FM by FS).

In all the experiments above there appeared a conflict between the two tasks assigned to the AV: when the moving object passes near an obstacle, the AV is not able to tell which is the object and which the obstacle and while the moving object may continue moving away from the AV, this will stay in front of the obstacle. However the AV will indeed avoid the collision with the obstacle. Actually the AV will avoid collisions with obstacles, but only if they do not enter or appeared in the blind spot of the AV's sonar units (0 to 40cm around the wheelchair).

CHAPTER X

CONCLUSIONS, RECOMMENDATIONS AND FUTURE WORK

Conclusions

The following conclusions and important results are obtained with the present research project:

1) A general structure to support the CAN in programs written for the ENAT board was developed and implemented.

2) The substitution of the M68HC11EVB boards by the ENAT boards, to enable the Propulsion and Vision Modules to connect to the CAN, was made successfully by implementing a set of necessary modifications in hardware and software.

3) The problem on how to communicate the PC (Supervisor Module), which does not have a CAN interface, with the ENAT boards (Vision and Propulsion Modules), was solved by implementing the Serial Communications Section and the gateway. To do this a serial communications protocol, based on the transport of ASCII characters, was designed and implemented to be used by both the PC and the ENAT board (gateway).

4) A Supervisor Module was designed and implemented. With the development of this module the following conclusions, regarding the implementation of the Fuzzy Logic Controller (FLC) can be made:

4.1) A change in one rule may affect the overall performance of the FLC. So it can be said that the more rules the harder the work is to modify or tune up the rule base. Furthermore, the fact pinpointed by Lee [12] that the problem of the interaction between rules is complex (and not as yet well understood) was confirmed in this research.

4.2) The ways the weighting factor or firing strength "fi" is calculated is not an influencing factor in the performance of the AV's FLC. Both the minimum operation and the algebraic product in the Cartesian Product led to the same AV's performance, in spite of the fact that the second one preserves the contribution of each input variable rather than the dominant one only.

4.3) It was proven that the use of variable supports for input and output fuzzy sets, along with the use of the "Quality Method" of Defuzzification, offers the best results regarding controller performance.

4.4) Another contribution of this research is the compactness, shortness and easiness in making modifications to the FLC software while tuning it up.

4.5) Changing the maximum values for the steering angle has a very little effect on the AV's performance.

5) The general objective of the present research, the design and integration of hardware and software to implement the AV, was achieved. And so implemented as a distributed system that uses the CAN, the AV gets as a special feature its "ability to grow" and therefore more microcontrollers can be added to make of the AV a robot that can perform more tasks and even more complex.

6) The following of a moving object and the avoidance of obstacle collisions were performed satisfactorily by the AV as independent activities. However the AV may "loose" the moving object when there is an obstacle in its neighborhood. The collision avoidance could not be implemented as planned, because three of the six sonars could not be used. Possible solutions for this are presented in the next section.

Recommendations and Future Work

1) The AV may be used as a tool to keep developing theoretical foundations in the analysis, design and use of distributed computer systems in automatic and intelligent control.

2) Based on conclusion 5) modules to implement voice commands and a "human-like-arm" movement module may be added to the AV. To do this the new messages and their PDUs have to be designed as explained in this research in order to connect to the CAN.

3) In order to avoid the "loosing" of the moving object when an obstacle is near a suggestion is as follows: According to Lee [12] the interactivity of rules can be controlled by the choice of fuzzy implication and type of composition. Lee suggests to use the concept of fuzzy clustering of fuzzy control rules to improve the consistency of rules, and recommends Sugeno's reasoning and identification algorithm to solve these problems.

4) To solve the problems of the sonar readings being affected by the motors and also to improve the AV's performance one or several of the following recommendations may be implemented:

4.1) To move the sonars 4, 5 and 6 far away enough from the motors.

4.2) To isolate the "big black box" from wheelchair structure.

4.3) To change floating ground by connecting all elements that use ground to one unique point or common ground.

4.4) To use optoisolators 4N26 or 4N35 to connect signals of sonar units to provide ground isolation.

4.5) To "shield" lines that carry computer pulses specially the ones going close to motors.

5) With the encountered problems solved (the input capture pins problem maybe solves with recommendation 4)), it is recommended to use Shanley's Supervisor with the suggested adjustments as shown in appendix F.

6) It has been recommended by Andujar [2] and Shanley [24] as future work, that in case of an emergency the Vision Module temporarily gain control of the Propulsion Module, that is, the Supervisor commands will be ignored until the emergency situation is taken care of by the Vision. The Vision Module should also monitor the Supervisor status or state in case of failure. If the Supervisor becomes inoperative the Vision should place the robot in a safe location and command the Propulsion to stop. In case of failure of the Supervisor or the Vision the Propulsion should stop immediately and wait for further instructions.

7) Also a new module may be added to the system to collect information from bump sensors to be placed in the blind spot of the sonars.

BIBLIOGRAPHY

1. Acuña, Enrique. "Conclusion of the Physical Design and Construction of the CPUCR (Central Processing Unit Costa Rica)". Licenciatura's Thesis, San José, Costa Rica:Universidad de Costa Rica, February, 1993.

2. Andujar, Ricardo. "Autonomous Vehicle Control Using Fuzzy Inference and a Fast Path Planning Algorithm". M. S. Thesis, Stillwater, OK, USA: Oklahoma State University, 1993.

3. Dearborn Group, Inc. Extended CAN Network Anlaysis Tool ENAT User's Manual Version 1.0. Michigan,USA: 1993.

4. Driankov, D., Hellendorn, H. and Reinfrank, M. An Introduction to Fuzzy Control. USA: Springer-Verlag, 1993.

5. Eccles, William. Programming the Motorola M68HC11. MA, USA: Addison-Wesley, 1992.

6. Ellington, M. and Steeger, L. "CFLOW". USA: 1985

7. El-Rewini, Hesham; Lewis, Ted and Shriver, Bruce. "Parallel and Distributed Systems From Theory to Practice." IEEE Parallel and Distributed Technology August, 1993.

8. Fowler, E. "Machine Intelligence Research Group." 7th Oklahoma Symposium on Artificial Intelligence. Oklahoma State University: November, 1993.

9. Gaudiano, Paolo; Zalama, Eduardo and López-Coronado, Juan. "A Real-Time, Unsupervised Neural Network for Low-Level Control of a Mobile Robot in a Nonstationary Environment." Neural Networks, Vol. 8: 1995.

10. Hellendorn, H and Thomas, C. "Defuzzification in Fuzzy Controllers." Journal of Intelligent and Fuzzy Systems, Vol. 1: 1993.

11. Intel Corporation. 82527 Serial Communications Controller, IL, USA: 1993.

12. Lee, C. "Fuzzy Logic in Control Systems: Fuzzy Logic Controller -Part I and Part II." IEEE Transactions on Systems, Man and Cybernetics. Vol. 20 (2), March/April, 1990.

13. Lee, Chun-Lin. "Ultrasonic Ranging System of an Obstacle-Avoidance Robot." M. S. Thesis, Stillwater, OK, USA: Oklahoma State University, 1993.

14. Motorola Inc. HC11 MC68HC11A8 Programming Reference Guide. 1990

15. Motoroal Inc. HC11 M68HC11 Reference Manual. 1991.

16. Motorola Inc. HC11 MC68HC11A8 Technical Data. 1991.

17. Motorola Inc. M68HC11EVB Evaluation Board User's Manual. 1986.

18. New-Micros, Inc. "Small C Compiler for 68HC11." TX, USA: 1993.

19. Newton, John. "Propulsion Design and Implementation for an Autonomous Vehicle." M. S. Thesis, Stillwater, OK, USA: Oklahoma State University, 1994.

20. Polaroid Inc. Ultrasonic Ranging System Description, Operation and Use Information for Conducting Tests and Experiments with Polaroid's Ultrasonic Ranging System Components. 1992.

21. Society of Automotive Engineers. Recommended Practice for Serial Control and Communications Network (Class C) for Truck and Bus Applications. SAE J1939: 1993.

22. Society of Automotive Engineers. "Data Link Layer-SAE J1939/21 Jul. 94." SAE Handbook. Vol. 2, Sections 15-26: 1995.

23. Society of Automotive Engineers. "Vehicle Application Layer-SAE J1939/71 Aug. 94." SAE Handbook. Vol. 2, Sections 15-26: 1995.

24. Shanley III, Robert Loren. "Environment Mapping and Adaptive Fuzzy Logic Control for an Autonomous Vehicle." M. S. Thesis, Stillwater, OK, USA: Oklahoma State University, 1994.

25. Simmons, Reid. "Structured Control for Autonomous Robots." IEEE Transactions on Robotics and Automation. Vol. 10 (1), February, 1994.

26. Texas Instruments, Inc. The TTL Data Book for Design Engineers. TX, USA: 1976.

27. Walker, Ian. "Book Review: An Introduction to Intelligent and Autonomous Control." IEEE Transactions on Robotics and Automation. Vol. 10 (1), February, 1994.

28. Wang, L. Adaptive Fuzzy Systems and Control Design and Stability Analysis. USA: Prentice-Hall, 1994.

29. Warner Books. Webster's New World Dictionary of the American Language. NY, USA: 1984.

30. Wilson, Gregory. "A Glossary of Parallel Computing Terminology." IEEE Parallel and Distributed Technology. February, 1993.

APPENDIXES

APPENDIX A

ROUTINES FOR CAN INITIALIZATION

```
#asm
*
* SUBROUTINES FOR CAN INITIALIZATION==========
*
* Starting up of CAN
*

INIT            LDAA #$41
                STAA CTLREG
                CLRA
                STAA RESETHI
                JSR DELAY
                STAA RESETLO
                JSR DELAY
                STAA RESETHI
                JSR DELAY
WRESET          LDAA CPUIFR     *LOAD CPU INTERFACE AND WAIT FOR RESET TO = 0
                ANDA #$80       *MASK RESET STATUS
                BNE WRESET
                JSR DELAY
                LDAA #$41
                STAA CTLREG
                LDAA #$40
                STAA CPUIFR

* end of starting of CAN

                RTS

*
* Initialization of CAN registers
*

INITC           LDAA #$20
                STAA $CA1F      *CLOCKOUT
                LDAA #$00
                STAA $CA2F      *BUSCON
                LDAA #$C1
                STAA $CA3F      *BUSTIM0
                LDAA #$C9
                STAA $CA4F      *BUSTIM1

*//////////////////////////////
* SET GLOBAL MASKS TO FF
*
                LDAA #$FF
                STAA $CA06      *STD1
                STAA $CA07      *STD0
                STAA $CA08      *EXT3
                STAA $CA09      *EXT2
                STAA $CA0A      *EXT1
                STAA $CA0B      *EXT0
                STAA $CA0C      *MSK3
                STAA $CA0D      *MSK2
                STAA $CA0E      *MSK1
```

```
                STAA $CA0F     *MSK0


*////////////////////////////////
* TURN OFF ALL CONTROL REGISTERS
*

                LDAA #$55
                STAA $CA10     *M1
                STAA $CA20     *M2
                STAA $CA30     *M3
                STAA $CA40     *M4
                STAA $CA50     *M5
                STAA $CA60     *M6
                STAA $CA70     *M7
                STAA $CA80     *M8
                STAA $CA90     *M9
                STAA $CAA0     *M10
                STAA $CAB0     *M11
                STAA $CAC0     *M12
                STAA $CAD0     *M13
                STAA $CAE0     *M14
                STAA $CAF0     *M15

* end of initialization of CAN registers

                RTS


*//////////////////////////
* CONFIGURE CAN PORTS
*

PORTS   LDAA #$FF
                STAA CPORT1    *SET UP PORT 1 AS OUTPUTS
                LDAA #$00
                STAA PORT1     *TURN OFF ALL LEDS
                LDAA #$00      *SET UP PORT 2 AS INPUTS
                STAA CPORT2
                RTS

*
* Delay of 20 ms
*

DELAY   PSHY
                LDY #6666
UP      DEY
                BNE UP
                PULY
                RTS

#endasm
```

```
/***********          */
/*  82527.H          */
/***********          */


#define MES_BASE 0xCA10
#define MESS1   MES_BASE+16*0
#define MESS2   MES_BASE+16*1
#define MESS3   MES_BASE+16*2
#define MESS4   MES_BASE+16*3
#define MESS5   MES_BASE+16*4
#define MESS6   MES_BASE+16*5
#define MESS7   MES_BASE+16*6
#define MESS8   MES_BASE+16*7
#define MESS9   MES_BASE+16*8
#define MESS10  MES_BASE+16*9
#define MESS11  MES_BASE+16*10
#define MESS12  MES_BASE+16*11
#define MESS13  MES_BASE+16*12
#define MESS14  MES_BASE+16*13
#define MESS15  MES_BASE+16*14

#define CONT0  0x000
#define CONT1  0x001
#define ARB0   0x002
#define ARB1   0x003
#define ARB2   0x004
#define ARB3   0x005
#define MCONF  0x006
#define DATA0  0x007
#define DATA1  0x008
#define DATA2  0x009
#define DATA3  0x00A
#define DATA4  0x00B
#define DATA5  0x00C
#define DATA6  0x00D
#define DATA7  0x00E

#define INT_REG 0xCA5F
#define PORT1  0xCADF
#define PORT2  0xCACF
```

APPENDIX B

VISION MODULE'S SOFTWARE

```
/*******************************************************
*    John Newton   MAE 5010   Robot Project
*          saved under sonar_sc.c
* Sonar Implementation with screen output
*             On the HC-11
*******************************************************/
/*
*        Modified by Enrique Acuña to be used in the
*        Controller Area Network (CAN), Dec. 1995
*        File name: SONAR1.C
*
*******************************************************/

/* Definition of starting addresses */

#code   0x20F0
#data   0x2000
#include <startup.c>

/* Including of routines for CAN initialization */

#include "can_ini.c"

/* Register bank mapping */

#define  REG_BASE 0x1000
#include <68hc11.h>
#include <82527.h>

/* File including section */

#include <stdio.h>
#include <delay.c>
#include <sonar.ini>                                  /* Variable Initialization */

/* Section for global variables and constants */

#define REQ_PGN_PF   0xEA  /* 234 */
#define Son1to4_PF   0xEB  /* 235 */
#define Son56_PF     0xE9  /* 233 */

/* Interrupt Service Routines */

interrupt toi()                                /* Timer Overflow Interrupt Routine */
{
        sys_clk++;
        /* Reset the timer overflow interrupt flag so it can happen again */
        bit_clr(REG_BASE+TFLG2,0x7F);
}

/* Receive routine: Here the typical receive routine recommended by the
SAE J1939 for Data Link Layer is implemented partially, because only
specific requests and PDU1 format (DA specific) are used */
```

```
interrupt irq()
{
d_int();
mess = MES_BASE + (peekb(INT_REG)-3)*16;
if (peekb(INT_REG)>0x02) {
                        pokeb(mess+CONT1,0xFD); /* clear NewDat */
                        PF = peek(mess+ARB0)/8;
/*                      DA = peek(mess+ARB1)/8;
                        SA = peek(mess+ARB2)/8;
*/                      if (PF==REQ_PGN_PF) {   /* is this a request of certain PGN? */
                                if (peekb(mess+DATA1)==Son1to4_PF) {
                                        upd3();
                                        send(MESS3);
                                }
                                if (peekb(mess+DATA1)==Son56_PF) {
                                        upd4();
                                        send(MESS4);
                                }
                        }
                }
        pokeb(mess+CONT0,0xFD);     /* to reset IntPnd */
        e_int();
}

main()
{

/* Section of initialization of CAN */

#asm
* can development for TX and RX
*
* ADDRESS CONSTANTS
*
RESETHI EQU $CC00
RESETLO EQU $CB00
CTLREG  EQU $CA00
CPORT1  EQU $CA9F
CPORT2  EQU $CAAF
PORT1   EQU $CADF
PORT2   EQU $CACF
CPUIFR  EQU $CA02
*
* START MAIN PROGRAM
*
* Initialize CAN
*
                LDS #$7FBF    * LOAD STACK
                SEI
                LDX $1000
                JSR INIT      * ROUTINE FOR STARTING UP OF CAN
                JSR INITC     * ROUTINE FOR INITIALIZING CAN REGISTERS
                JSR PORTS
                LDAA #$00     *CLEAR STATUS REGISTER
```

```
                STAA $CA01
                LDAA #$02       *ENABLE INTERRPUTS ON CAN
                STAA $CA00
#endasm

        poke(0xD2,toi);                         /* Timer Overflow Interrupt */
        bit_set(REG_BASE+TFLG2,0x80);           /* Clear the Timer Overflow Flag */
        bit_set(REG_BASE+TMSK2,0x80);           /* Enable the Timer Overflow
                                                   Interrupt*/

/* Section for setting or configuring the message objects to be used */

set_mes1();
set_mes3();
set_mes4();

poke(0xF0,irq);             /*IRQ via IQR pin */
var_ini();
e_int();

        while (1)
        {
                unsigned int i;
                i=0;
                while (i < 10*(255-peekb(0x20)))
                        i++;
                count++;
                if (count == 10)
                {
                        sam_freq=3050/sys_clk; /* Maximum Sampling Frequency = 3.4 Hz */
/*                      fprintf(stdout,"Sampling Frequency X10 (Hz): %u\n",sam_freq);
*/                      count=0;
                        sys_clk=0;
                }
                dist1 = range(1,67,67);
                dist2 = range(2,67,67);
                dist3 = range(4,67,67);
                dist4 = range(8,67,67);
                dist5 = range(16,67,67);
                dist6 = range(32,67,67);
        }
}

range(out_bit,time_up,delay_time)           /* 1/time_up and 1/delay_time where
                                               time_up and delay_time are in milli-
                                               seconds. */
unsigned int out_bit,time_up,delay_time;
{
        unsigned int dist,i;
        num=0;
        max_num=8490/time_up;
        max_i=57598/delay_time;
        pokeb(REG_BASE+PORTB,out_bit);
        /* sam freq is approximately 8490 Hz */
```

```
        while (peekb(REG_BASE+PORTE) == 0 && num < max_num)
        {
                num++;
        }
        i=0;
        /* sam freq is approx 57598 Hz */
        while(i < max_i)
                i++;
        pokeb(REG_BASE+PORTB,0x00);

        if (num < 48)
                dist=num*2;
        else if ((num > 49) && (num < 60))
                dist=num*2+5;
        else if ((num > 61) && (num < 71))
                dist=num*2+7;
        else if ((num > 72) && (num < 105))
                dist=num*2+12;
        else
                dist=num*2+17;
 return dist;
}

/* set message objetct #1 for receiving request of Son1-4 from Super.*/

set_mes1() {
        pokeb(MESS1+CONT0,0x55);
        pokeb(MESS1+CONT1,0x55);
        pokeb(MESS1+ARB0,0xC7);
        pokeb(MESS1+ARB1,0x50);
        pokeb(MESS1+ARB2,0x10);
        pokeb(MESS1+ARB3,0x00);
        pokeb(MESS1+MCONF,0x84);
        pokeb(MESS1+CONT0,0x99);
}

/* set message object #3 for transmiting response to request of
        Son1-4 from Super */

set_mes3() {
        pokeb(MESS3+CONT0,0x55);
        pokeb(MESS3+CONT1,0x59);
        pokeb(MESS3+ARB0,0xC7);
        pokeb(MESS3+ARB1,0x58);
        pokeb(MESS3+ARB2,0x00);
        pokeb(MESS3+ARB3,0x10);
        pokeb(MESS3+MCONF,0x8C);
        pokeb(MESS3+CONT0,0xBF);
}

/* set message objetct #4 for transmiting response to request of
        Son5-6 from Super. */

set_mes4() {
```

```
        pokeb(MESS4+CONT0,0x55);
        pokeb(MESS4+CONT1,0x59);
        pokeb(MESS4+ARB0,0xC7);
        pokeb(MESS4+ARB1,0x48);
        pokeb(MESS4+ARB2,0x00);
        pokeb(MESS4+ARB3,0x10);
        pokeb(MESS4+MCONF,0x8C);
        pokeb(MESS4+CONT0,0xBF);
}

/* ////////////////
* Update data in message object 3
*/
upd3() {

        pokeb(MESS3+CONT1,0xFA);  /*START OF UPDATING (CPUUpd=NewDat=1)*/

        poke(MESS3+DATA0,dist1); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        poke(MESS3+DATA2,dist2);
        poke(MESS3+DATA4,dist3);
        poke(MESS3+DATA6,dist4);

        pokeb(MESS3+CONT1,0xF7);   /*END OF DATA UPDATING (CPUUpd=0)*/
 }
/* ////////////////
* Update data in message object 4
*/
upd4() {

        pokeb(MESS4+CONT1,0xFA);  /*START OF UPDATING (CPUUpd=NewDat=1)*/

        poke(MESS4+DATA0,dist5); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        poke(MESS4+DATA2,dist6);

        pokeb(MESS4+CONT1,0xF7);   /*END OF DATA UPDATING (CPUUpd=0)*/
 }
/*
* MESSAGE OBJECT SENDS ITS DATA FRAME
*/
send(mess)
unsigned int mess;
{
/*TXRQUST=1 CPPUUD=0 NEWDAT=1*/
pokeb(mess+CONT1,0xE6);
}
```

```
/*********************************************************
*    John Newton   MAE 5010   Robot Project       *
*           saved under sonar.ini            *
*  Sonar Implementation variable initialization    *
*              On the HC-11                 *
*********************************************************/
/*
*        Modified to be used in the Controller Area Network (CAN)
*        by: Enrique M. Acuña, Dec. 1995
*
*********************************************************/

unsigned char PF,DA,SA;
unsigned int mess;
unsigned int temp,temp1,sys_clk;
unsigned int count,sam_freq;
unsigned int dist1,dist2,dist3,dist4,dist5,dist6;
unsigned int max_num,max_i,num;
var_ini()
{
        sam_freq=0;
        sys_clk=0;
        count=0;
        num=0;
        dist1=0;
        dist2=0;
        dist3=0;
        dist4=0;
        dist5=0;
        dist6=0;
}
```

APPENDIX C

PROPULSION MODULE'S SOFTWARE

```
/*********************************************************
 *    John Newton   MAE 5010   Robot Project
 *          saved under DRIVE2_s.c
 *     Implementation with screen output
 *             On the HC-11
 *********************************************************/
/*
 *        Modified to be used in the Controller Area Network (CAN)
 *        by: Enrique M. Acuña, Dec. 1995
 *        File name: DRIVE4.C
 *
 *********************************************************/

/* Definition of starting addresses */

#code   0x20F0
#data   0x2000
#include <startup.c>

/* Including of routines for CAN initialization */

#include "can_ini.c"

/* Register bank mapping */

#define  REG_BASE 0x1000
#include <68hc11.h>
#include <82527.h>

/* File including section */

#include <stdio.h>
#include <delay.c>

/* Link in the device driver for the Motorola 68HC11 EVB */

#include "drive.ini"        /* Variable Initialization */
#include "hc11fp1.c"        /* Floating Point Emulation Library */
#include "encode3.c"        /* Transforms Encoder Data into X and Y position */
#include "look_tab.h"       /* Look-up table for sin and cos */
                            /* Also sets control gains and saturation voltage */

/* Link in the following libraries (nearly all needed by fprintf() )*/

#include <fopen.c>
#include <fputs.c>
#include <fputc.c>
#include <itoa.c>
#include <atoi.c>
#include <is.c>
#include <fprintf.c>
#include <itoab.c>
#include <reverse.c>
#include <strlen.c>
```

```
#include <EVB_out.c>

/* Section for global variables and constants */

#define REQ_PGN_PF  0xEA  /* 234 */
#define RS_C_id_PF   0xED  /* 237 */
#define XYpos_PF     0xEC  /* 236 */

FILE        stdout;
unsigned int a_v_temp;
unsigned int t_d_temp;
unsigned int xi,yi,xs,ys;

/* Interrupt Service Routines */

/* Called whenever the system timer overflows (about every 0.03 seconds) */

interrupt toi()                         /* Timer Overflow Interrupt Routine */
{
        toi_flag=0;
        loop++;
        sys_clk++;
        /* Reset the timer overflow interrupt flag so it can happen again */
        bit_set(REG_BASE+TFLG2,0x80);

        if (loop == 31)  /* one second has transpired if loop = 31 */
        {
                time++;
                loop=0;
/*              fprintf( stdout,"num: %u \n",num);
*/              num=0;

/* following lines to test XY position */

                if (x_pos < (offset-1))
                {
                        xi=offset-x_pos;
                        xs=0;
                        }
                else
                {
                        xi=x_pos-offset;
                        xs=1;
                }
                if (y_pos < (offset-1))
                {
                        yi=offset-y_pos;
                        ys=0;
                        }
                else
                {
                        yi=y_pos-offset;
                        ys=1;
```

```
                }
/*              fprintf( stdout,"ld: %u rd: %u \n",left_inc-offset,right_inc-offset);
*/              fprintf( stdout,"x%u: %d y%u: %d t: %u rd: %u \n",xs,xi,ys,
                                                yi,ang-offset,right_disp-offset);

/*              fprintf( stdout,"des_vel: %u  \n",des_vel-offset);
                fprintf( stdout,"des_phi: %u  \n",des_phi-offset);
*/
        }
}

interrupt lt()      /* Interrupt takes place when an encoder pulse is detected */
{
        left_inc++;                                     /* Increment the left wheel position */
/*      bit_clr(REG_BASE+TFLG1,0xFB);
*/      bit_set(REG_BASE+TFLG1,0x04);
}
interrupt rt()
{
        right_inc++;    /* Increment the right wheel position */
/*      bit_clr(REG_BASE+TFLG1,0xFD);
*/      bit_set(REG_BASE+TFLG1,0x02);
}

/* Receive routine: Here the typical receive routine recommended by the
SAE J1939 for Data Link Layer is implemented partially, because only
specific requests and PDU1 format (DA specific) are used */

interrupt irq()
{
d_int();
mess = MES_BASE + (peekb(INT_REG)-3)*16;
if (peekb(INT_REG)>0x02)  {
                        pokeb(mess+CONT1,0xFD);  /* clear NewDat */
                        PF = peek(mess+ARB0)/8;
/*                      DA = peek(mess+ARB1)/8;
                        SA = peek(mess+ARB2)/8;
*/                      if (PF==REQ_PGN_PF) {
                                        if (peekb(mess+DATA1)==RS_C_td_PF) {
                                                upd5();
                                                send(MESS5);
                                        }
                                        if (peekb(mess+DATA1)==XYpos_PF) {
                                                upd6();
                                                send(MESS6);

/* this part to correct XY position and heading angle to avoid effect of floating point unavailability */

                                                x10_pos=offset;
                                                y10_pos=offset;
/*                                              theta=offset;                   */

                                        }
                        }
```

```
                        if (mess==MESS1) {
                                        c_des_vel=peek(MESS1+DATA0);
                                        c_des_phi=peek(MESS1+DATA2);
                                        can_mot=peek(MESS1+DATA4);
                                        upd4();
                                        send(MESS4);
                        }
        }
        pokeb(mess+CONT0,0xFD);     /* to reset IntPnd */
        e_int();
}

main()
{

/* Section of initialization of CAN */

#asm
*can development for TX and RX
*
* ADDRESS CONSTANTS
*
RESETHI EQU $CC00
RESETLO EQU $CB00
CTLREG  EQU $CA00
CPORT1  EQU $CA9F
CPORT2  EQU $CAAF
PORT1   EQU $CADF
PORT2   EQU $CACF
CPUIFR  EQU $CA02
*
* START MAIN PROGRAM
*
* Initialize CAN
*
                LDS #$7FBF    * LOAD STACK
                SEI
                LDX $1000
                JSR INIT      * ROUTINE FOR STARTING UP OF CAN
                JSR INITC     * ROUTINE FOR INITIALIZING CAN REGISTERS
                JSR PORTS
                LDAA #$00     *CLEAR STATUS REGISTER
                STAA $CA01
                LDAA #$02     *ENABLE INTERRPUTS ON CAN
                STAA $CA00
#endasm

        poke(0xD2,toi);         /* Timer Overflow Interrupt */
        poke(0xEA,lt);          /* Use PA2/IC1 for left position */
        poke(0xE7,rt);          /* Use PA1/IC2 for right position */

        bit_set(REG_BASE+TFLG2,0x80);       /* Clear the Timer Overflow Flag */
        bit_set(REG_BASE+TMSK2,0x80);       /* Enable the Timer Overflow Interrupt */
```

```
        bit_set(REG_BASE+TMSK1,0x06);       /* Set up the Input Capture Pins */
        bit_set(REG_BASE+TCTL2,0x14);       /* to detect the encoder pulses */
        bit_set(REG_BASE+TFLG1,0x06);       /* from each encoder */

        bit_clr(REG_BASE+DDRC,0xE0);/* Change the data direction on Port C */
                                            /* to make PC7,PC6 and PC5 inputs */
        bit_set(REG_BASE+PORTA,0x30);       /* CLK signals on registers to D/A
                                            converters are high */

/* Section for setting or configuring the message objects to be used */

set_mes1();
set_mes2();
set_mes4();
set_mes5();
set_mes6();

poke(0xF0,irq);  /*IRQ via IQR pin */

        look_tab();     /* Install the Look-Up table into RAM */
        var_ini();      /* Initialize Variables */
e_int();
        stdout=fopen(EVB_out);
        fprintf(stdout,"Starting system timer...\n");

/*
* START LOOP
*/

        while (1)
                main_loop();

}

main_loop()
{
        num++;
        rd_anlg();              /* Read Analog Data (Wheel Speeds) */
        super_in();             /* Obtain des_vel, des_phi, and
                                mot_onoff from supervisory input */
        if ((mot_onoff == 0)    /* mot_onoff = 0 if Emergency stop is */
        || (can_mot == 0))      /* needed */
        {
                des_phi=offset;
                des_vel=offset;
                bit_clr(REG_BASE+PORTA,0x40);       /* DeActivate the relay which connects
                                                    the computer's D/A converters to the
                                                    wheelchair electronics */
        }
        else if ((mot_onoff == 1)       /* mot_onoff = 1 if Emergency stop is */
        & (can_mot == 1))               /*       not needed. */
        {
                bit_set(REG_BASE+PORTA,0x40);       /* Activate the relay which connects
                                                    the computer's D/A converters to the
```

```
                                                        wheelchair electronics */
                des_vel=c_des_vel;
                des_phi=c_des_phi;
        }
        /* a_v=(left_vel+right_vel)/2 */
        temp=add(shift_left(left_vel,10),shift_left(right_vel,10)); /* X10 */
        a_v=div(temp,32769);  /* X10 */

        /* t_d=(right_vel-left_vel)*r*2/l; */
        temp=sub(shift_left(right_vel,10),shift_left(left_vel,10));
        t_d=shift_left(mult(temp,32776),10);     /* X10 */

/**** des_vel and des_phi should have a bios of offset but not X10 ****/
        if (des_phi > (32857+1))            /* Limits des_phi to +/- 90*/
        des_phi = 32857;            /* degrees */
        if (des_phi < (32677-1))
        des_phi = 32677;
        if (des_vel > (32809+1))            /*  Limits des_vel to +/- 42 rpm or
                                            3 feet per seconds */
        des_vel = 32809;
        if (des_vel < (32725-1))
        des_vel = 32725;

/* des_a_v=des_vel*cos(des_phi*pi/180); */
        temp=cos_look(des_phi);
        des_a_v=shift_left(mult(des_vel,shift_left(temp,10)),10); /* X10 */

 /* des_t_d=des_vel*sin(des_phi*314/18000)*r/h; */
        temp=sin_look(des_phi);
        temp1=mult(des_vel,shift_left(temp,10));  /* X100 */
        des_t_d=shift_left(mult(temp1,4+offset),100);  /* X10 */

/* t_d_er=-1*(t_d-des_t_d); */
        t_d_er=sub(t_d,des_t_d);  /*  X10 */

/* a_v_er=-1*(a_v-des_a_v); */
        a_v_er=sub(des_a_v,a_v);     /* X10 */

/* a_v_er_sum+=(a_v_er*sam_time); */
        a_v_temp=shift_left(mult(a_v_er,32770),100);
        temp1=add(a_v_temp,a_v_er_sum);
        a_v_er_sum=temp1;                       /* X10 */

/* t_d_er_sum+=(t_d_er*sam_time); */
        t_d_temp=shift_left(mult(t_d_er,32770),100);
        temp1=add(t_d_temp,t_d_er_sum);
        t_d_er_sum=temp1;                       /* X10 */


/* vel_out=(prop_gain*a_v_er+int_gain*a_v_er_sum); */
        temp=shift_left(mult(int_gain,a_v_er_sum),10);  /* X10 */
        temp1=shift_left(mult(prop_gain,a_v_er),100);  /* X10 */
        vel_out=add(temp,temp1);
```

```
        if (vel_out>(offset+u_max))         /* Anti-Windup routine for avg_vel */
        {
                vel_out=offset+u_max;     /* Set vel_out to max voltage */
                temp1=sub(a_v_er_sum,a_v_temp);    /* Turn off integration */
                a_v_er_sum=temp1;
        }
        if (vel_out<(offset-u_max))
        {
                vel_out=offset-u_max;
                temp1=sub(a_v_er_sum,a_v_temp);
                a_v_er_sum=temp1;
        }

/* theta_out=-(prop_gain*t_d_er+int_gain*t_d_er_sum)*19/44; */
        temp=shift_left(mult(theta_int,t_d_er_sum),10); /* X10 */
        temp1=shift_left(mult(theta_prop,t_d_er),100); /* X10 */
        theta_out=add(temp,temp1);

        if (theta_out>(offset+u_max))                /* Anti-Windup for theta_out */
        {
                theta_out=offset+u_max;
                temp1=sub(t_d_er_sum,t_d_temp);
                t_d_er_sum=temp1;
        }
        if (theta_out<(offset-u_max))
        {
                theta_out=offset-u_max;
                temp1=sub(t_d_er_sum,t_d_temp);
                t_d_er_sum=temp1;
        }
        encode();                   /* Transforms Encoder Data into X and Y position */
        write_da();   /* Write to D/A converters */
/*Use the Timer OverFlow Interrupt to sample at 30.5Hz */
        toi_flag=1;
        while(toi_flag==1);        /* Pole for toi_flag=0 */
/*      delay(31);
*/
}


rd_anlg()                           /* Read Analog Data (wheel speed) */
{

        pokeb(REG_BASE+ADCTL,0x14);   /* Prepare A/D's for input */
/*      right_vel=mult(vel_conv,peekb(REG_BASE+ADR1)+32640);          /* X100 */
/*      left_vel=mult(vel_conv,peekb(REG_BASE+ADR2)+32642);           /* X100 */

        right_vel=mult(vel_conv,peekb(REG_BASE+ADR1)+offset-rt_zero);          /* X100 */
        left_vel=mult(vel_conv,peekb(REG_BASE+ADR2)+offset-lt_zero);           /* X100 */

}

super_in()                          /* This function should contain the commands necessary
                                    to obtain des_vel, des_phi, and mot_onoff from
```

```
                                the supervisor. */
{
        /* Desired Velocity and Steering Angle is read in here through the A/D
        converter.  The units are wheel RPM for desired velocity and degrees,
        varing from -90 to 90, for steering angle.  Multiply desired velocity in
        feet per second by 14.21 to get RPM. */
/*      des_vel=(255-peekb(REG_BASE+ADR3))/3+offset-42; /* Read des_vel from A/D

                                                        pin */

/*      des_phi=peekb(REG_BASE+ADR4)+32640;     /* Read des_phi from A/D
                                                    pin */
/*      temp=254 | (peekb(REG_BASE+PORTC)>>5);  /* Reads status of Digital Input Pin PC5
                                                    to determine if an emergency stop
                                                    is needed. */
/*      mot_onoff=temp-254;
*/
        if ((peekb(REG_BASE+PORTC)&0x20)==0x20)
                mot_onoff=1;
        else
                mot_onoff=0;
}

write_da()              /* Write to D/A converters */
{
/* vel_out should be given to this function as X10 + offset */

/* D/A Calculation - bits=6.8*volts + 60.2
(Voltage w/ respect to -4.5 volts from computer ground.) */

        temp=shift_left(mult(vel_out,offset+68),10);    /* X10 */
        temp1=shift_left(add(temp,offset+602),10);      /* X1 */
        if (temp1 < offset)
                temp1=offset;
        temp=temp1-offset;
        pokeb(REG_BASE+PORTB,temp); /* Write temp to port B from where the D/A
                                       converter receives forward/reverse voltage */
        bit_clr(REG_BASE+PORTA,0x10); /* A raising edge is produced on PA4
                                      for CLK signal on register to D/A
                                      converter for forward/reverse voltage */
        bit_set(REG_BASE+PORTA,0x10); /* CLK signal on register to D/A
                                      converter for forw/rev vol. is low */

/* theta_out should be returned to this function as X10 + offset */
/* D/A Calculation - bits=13.4*volts + 61.5
(Voltage w/ respect to -4.5 volts from wheelchair ground.) */

        temp=shift_left(mult(theta_out,offset+134),10);     /* X10 */
        temp1=shift_left(add(temp,offset+615),10);      /* X1 */
        if (temp1 < offset)
                temp1=offset;
        temp=temp1-offset;
        pokeb(REG_BASE+PORTB,temp);   /* Write temp to port B where the D/A
                                      conversion takes place for steering angle */
```

```
        bit_clr(REG_BASE+PORTA,0x20);  /* A raising edge on PA5 for CLK signal on
                                          resgister to D/A for steering angle is produced */
        bit_set(REG_BASE+PORTA,0x20); /* CLK signal on register to D/A
                                          converter for steering angle is low */
}


/* set message objetct #1 for receiving command with desired speed and
        steering angle from Super.*/

set_mes1()
{
        pokeb(MESS1+CONT0,0x55);
        pokeb(MESS1+CONT1,0x55);
        pokeb(MESS1+ARB0,0xC7);
        pokeb(MESS1+ARB1,0x70);
        pokeb(MESS1+ARB2,0x08);
        pokeb(MESS1+ARB3,0x00);
        pokeb(MESS1+MCONF,0x84);
        pokeb(MESS1+CONT0,0x99);
}

/* set message objetct #2 for receiving the requests of RS_C_td
        and XYpos from Super.*/

set_mes2()
{
        pokeb(MESS2+CONT0,0x55);
        pokeb(MESS2+CONT1,0x55);
        pokeb(MESS2+ARB0,0xC7);
        pokeb(MESS2+ARB1,0x50);
        pokeb(MESS2+ARB2,0x08);
        pokeb(MESS2+ARB3,0x00);
        pokeb(MESS2+MCONF,0x84);
        pokeb(MESS2+CONT0,0x99);
}

/* set message objetct #4 for transmiting ACK of command (with
        desired speed and steering angle) from Super. */

set_mes4() {
        pokeb(MESS4+CONT0,0x55);
        pokeb(MESS4+CONT1,0x59);
        pokeb(MESS4+ARB0,0xC7);
        pokeb(MESS4+ARB1,0x40);
        pokeb(MESS4+ARB2,0x00);
        pokeb(MESS4+ARB3,0x08);
        pokeb(MESS4+MCONF,0x8C);
        pokeb(MESS4+CONT0,0xBF);
}

/* set message objetct #5 for transmiting requested data (RS_C_td)
        to Super. */
```

```
set_mes5() {
        pokeb(MESS5+CONT0,0x55);
        pokeb(MESS5+CONT1,0x59);
        pokeb(MESS5+ARB0,0xC7);
        pokeb(MESS5+ARB1,0x68);
        pokeb(MESS5+ARB2,0x00);
        pokeb(MESS5+ARB3,0x08);
        pokeb(MESS5+MCONF,0x8C);
        pokeb(MESS5+CONT0,0xBF);
}
/* set message objetct #6 for transmiting data (XYposition) to Super. */

set_mes6() {
        pokeb(MESS6+CONT0,0x55);
        pokeb(MESS6+CONT1,0x59);
        pokeb(MESS6+ARB0,0xC7);
        pokeb(MESS6+ARB1,0x60);
        pokeb(MESS6+ARB2,0x00);
        pokeb(MESS6+ARB3,0x08);
        pokeb(MESS6+MCONF,0x8C);
        pokeb(MESS6+CONT0,0xBF);
}

/* ////////////////
* Update data in message object 4
*/
upd4() {

        pokeb(MESS4+CONT1,0xFA);  /*START OF UPDATING (CPUUpd=NewDat=1)*/

        pokeb(MESS4+DATA0,0x00); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        pokeb(MESS4+DATA1,0xFF);
        pokeb(MESS4+DATA2,0xFF);
        pokeb(MESS4+DATA3,0xFF);
        pokeb(MESS4+DATA4,0xFF);
        pokeb(MESS4+DATA5,0x00);
        pokeb(MESS4+DATA6,0xEE);
        pokeb(MESS4+DATA7,0x01);

        pokeb(MESS4+CONT1,0xF7);   /*END OF DATA UPDATING (CPUUpd=0)*/
}

/* ////////////////
* Update data in message object 5
*/
upd5() {

        pokeb(MESS5+CONT1,0xFA);  /*START OF UPDATING (CPUUpd=NewDat=1)*/

        poke(MESS5+DATA0,a_v); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        poke(MESS5+DATA2,angle);
        poke(MESS5+DATA4,t_d);
/*      poke(MESS5+DATA6,Brake);
*/
```

```
        pokeb(MESS5+CONT1,0xF7);    /*END OF DATA UPDATING (CPUUpd=0)*/
}
/* ////////////////
* Update data in message object 8
*/
upd6() {

        pokeb(MESS6+CONT1,0xFA);  /*START OF UPDATING (CPUUpd=NewDat=1)*/

        poke(MESS6+DATA0,x_pos); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        poke(MESS6+DATA2,y_pos);

        pokeb(MESS6+CONT1,0xF7);    /*END OF DATA UPDATING (CPUUpd=0)*/
}
/*
* MESSAGE OBJECT 4 SENDS ITS DATA FRAME
*/
send(mess)
unsigned int mess;
{
/*TXRQUST=1 CPPUUD=0 NEWDAT=1*/
pokeb(mess+CONT1,0xE6);
}



/*********************************************************
 *    John Newton   MAE 5010   Robot Project       *
 *           saved under drive.ini             *
 *         Velocity Control of Wheelchair        *
 *              On the HC-11                *
 *********************************************************/
/*
*       Modified to be used in Controller Area Network (CAN)
*       by: Enrique M. Acuña, Dec. 1995
*/
/*********PARAMETERS PROVIDED BY SUPERVISOR INPUT ************************/
unsigned int des_vel;                   /* Desired Velocity in rpm x1 with offset */
unsigned int des_phi;                   /* Desired Steering angle in degrees x1
                                        with offset */
unsigned int c_des_vel;                 /* Desired Velocity in rpm x1 with offset
                                        from CAN*/
unsigned int c_des_phi;                 /* Desired Steering angle in degrees x1
                                        with offset from CAN*/
unsigned int can_mot;                   /* emergency stop from CAN */
unsigned int mot_onoff;                 /* Governs whether the D/A converters
                                        are connected to the DC motors of the
                                        wheelchair. Provides emergency stop
                                        capability. 1=true and 0=false
                                        No Offset */
/*************************************************************************/

/*********PARAMETERS NEEDED BY SUPERVISOR ********************************/
unsigned int x_pos;             /* X position in inches x1 with offset  */
```

```
unsigned int y_pos;             /* Y position in inches x1 with offset*/
/*************************************************************************/

unsigned char PF,DA,SA;         /* PDU's corresponding parts for PF, DA and SA */
unsigned int mess;              /* contains the message object number */
unsigned int temp,temp1,temp2;  /* Temporary storage variables */
unsigned int sys_clk;           /* System Clock */
unsigned int toi_flag;          /* Timer Overflow Flag */
unsigned int loop;              /* Counts the number of timer overflows
                                   it takes for one second to go by */
unsigned int num;               /* Count the sampling rate */
unsigned int left_inc;          /* Displacement of left wheel during the */
                                /* time it takes to sample once. */
unsigned int right_inc;         /* Same as left_inc */
unsigned int left_vel;          /* Velocity of left wheel in rpm x100 */
unsigned int right_vel;         /* Velocity of right wheel in rpm x 100 */
unsigned int vel_conv;          /* Conversion from voltage to rpm */
unsigned int time;              /* Time in seconds */
unsigned int vel_out;           /* Forward Reverse Joystick Voltage */
unsigned int theta_out;         /* Side to Side joystick voltage */
unsigned int u_max;             /* Max joystick voltage */
unsigned int a_v;               /* Average Velocity x10 in rpm */
unsigned int t_d;               /* Theta dot in rpm x10 */
unsigned int des_a_v;           /* Desired Average Velocity x10 in rpm */
unsigned int des_t_d;           /* Desired Theta dot in rpm x10 */
unsigned int t_d_er;            /* Error in Theta dot */
unsigned int a_v_er;            /* Error in average velocity */
unsigned int a_v_er_sum;        /* Integration of average velocity error */
unsigned int t_d_er_sum;        /* Integration of theta dot error      */
unsigned int prop_gain;         /* Proportional gain read from
                                   look-up-table */
unsigned int int_gain;          /* Integral Gain read from look-up-table */
unsigned int theta_prop;        /* Proportional gain read from
                                   look-up-table */
unsigned int theta_int;         /* Integral Gain read from look-up-table */
unsigned int offset;            /* Half of 16 bits or 32767 */
unsigned int in_p_p;            /* Inches per Pulse */
unsigned int left_pos;          /* Same as left_inc only not in
                                   interrupt routine */
unsigned int right_pos;         /* Same as right_inc only not in
                                   interrupt routine */
unsigned int x_inc;             /* X position Increment */
unsigned int y_inc;             /* Y position Increment */
unsigned int x10_pos;           /* X position in inches x10 with offset */
unsigned int y10_pos;           /* Y position in inches x10 with offset */
unsigned int theta;             /* Heading Angle (degrees) x100. Varies
                                   from -180 to 180 degrees */
unsigned int angle;             /* Heading angle (degrees) x1 with offset
                                   Varies from -180 to 180 degrees */

/* variables used to make tests to solve the XY-position and heading angle problem */

unsigned int ang;
```

```
unsigned int rt_zero;
unsigned int lt_zero;

unsigned int left_disp;
unsigned int right_disp;
unsigned int reset_count;

var_ini()                                          /* Variable initialization */
{
        reset_count=0;
        offset=32767;
        theta=offset;                                          /* Zero variable */
        left_disp=offset;
        right_disp=offset;
        x10_pos=offset;          /* Zero variable */
        y10_pos=offset;          /* Zero variable */
        right_inc=offset;        /* Zero variable */
        p2=0xAAAA;
        p3=0xBBBB;
        left_inc=offset;         /* Zero variable */
        p1=0xAAAA;
        vel_conv=78+offset;       /* 100 X .78 RPM per bit */
        in_p_p=2+offset; /*       X10 X.2245 inches per pulse */
/* r=8.06 inches;      */
/* l=18.53 inches;     */
/* h=18.88 inches;     */
/*      prop_gain=peekb(0x20)+offset;       /* X100  Read from look_tab */
        prop_gain=5+offset;       /* X100  Read from look_tab */
/*      int_gain=peekb(0x21)+offset;      /* X10  Read from look_tab */
        int_gain=3+offset;      /* X10  Read from look_tab */
/*      theta_prop=peekb(0x23)+offset;       /* X100  Read from look_tab */
        theta_prop=3+offset;       /* X100  Read from look_tab */
/*      theta_int=peekb(0x24)+offset;      /* X10  Read from look_tab */
        theta_int=1+offset;      /* X10  Read from look_tab */
        a_v_er_sum=offset;                                     /* Zero variable */
        t_d_er_sum=offset;         /* Zero variable */
        des_phi=offset;          /* Zero variable */
/*      u_max=peekb(0x22);         /* Max joystick voltage x 10 read from
                                   look_tab */
        u_max=70;
        vel_out=offset;          /* Zero variable */
        theta_out=offset;         /* Zero variable */
        write_da();              /* Stop Motors */

        pokeb(REG_BASE+ADCTL,0x14);   /* Prepare A/D's for input */
        rt_zero=peekb(REG_BASE+ADR1);
        lt_zero=peekb(REG_BASE+ADR2);

}
```

```
/*********************************************************
 *    John Newton   MAE 5010   Robot Project       *
 *           saved under hc11fp.c              *
 *         Floating Point Emulation Library       *
 *              On the HC-11                *
 *********************************************************/
/*
*         Modified to overcome the Small C compiler error of the
*         inequalities
*         by: Enrique M. Acuña, Nov. 1995
*         File name: hc11fp1.c
*/
/*****************************************************************/
/*  Provides a means of having negative numbers            */
/*  when all variables have been declared unsigned         */
/*                                                         */
/*  Performs addition, multiplication, subtraction,        */
/*  division, and scaling by orders of magnitude.          */
/*  Also reads sine and cosine of -90 to 90 degrees from   */
/*  look-up-table.                                         */
/*****************************************************************/


unsigned int sign_one,sign_two;

check_sign(in_one,in_two)
unsigned int in_one,in_two;
{
        sign_one=1;
        sign_two=1;
if (in_one < (offset-1))
                sign_one=0;
if (in_two < (offset-1))
                sign_two=0;
}

mult(in_one,in_two)
unsigned int in_one,in_two;
{
        unsigned int out;
        check_sign(in_one,in_two);
        if (sign_one == 1 && sign_two == 1)
                out=offset+((in_one-offset)*(in_two-offset));
        else if (sign_one == 0 && sign_two == 0)
                out=offset+((offset-in_one)*(offset-in_two));
        else if (sign_one == 1 && sign_two == 0)
                out=offset-((in_one-offset)*(offset-in_two));
        else if (sign_one == 0 && sign_two == 1)
                out=offset-((offset-in_one)*(in_two-offset));
}

div(in_one,in_two)
```

```
unsigned int in_one,in_two;
{
        unsigned int out;
        check_sign(in_one,in_two);
        if (sign_one == 1 && sign_two == 1)
                out=offset+((in_one-offset)/(in_two-offset));
        else if (sign_one == 0 && sign_two == 0)
                out=offset+((offset-in_one)/(offset-in_two));
        else if (sign_one == 1 && sign_two == 0)
                out=offset-((in_one-offset)/(offset-in_two));
        else if (sign_one == 0 && sign_two == 1)
                out=offset-((offset-in_one)/(in_two-offset));
}

add(in_one,in_two)
unsigned int in_one,in_two;
{
        unsigned int out;
        check_sign(in_one,in_two);
        if (sign_one == 1 && sign_two == 1)
                out=offset+((in_one-offset)+(in_two-offset));
        else if (sign_one == 0 && sign_two == 0)
                out=offset-((offset-in_one)+(offset-in_two));
        else if (sign_one == 1 && sign_two == 0)
                out=offset+((in_one-offset)-(offset-in_two));
        else if (sign_one == 0 && sign_two == 1)
                out=offset+((in_two-offset)-(offset-in_one));
}

sub(in_one,in_two)
unsigned int in_one,in_two;
{
        unsigned int out;
        check_sign(in_one,in_two);
        if (sign_two == 1)
                out=add(in_one,offset-(in_two-offset));
        else
                out=add(in_one,offset+(offset-in_two));
}

shift_left(in_one,in_two)
unsigned int in_one,in_two;
{
        unsigned int out;
        if (in_one < (offset-1))
                out=offset-(offset-in_one)/(in_two);
        else
                out=(in_one-offset)/(in_two)+offset;

}

cos_look(in_one)   /* Provides cosine of in_one from look-up-table */
unsigned int in_one;     /* in_one should be in degrees x1 with offset */
                         /* -90 to 90 degrees only */
```

```
{
        unsigned int out;
        if (in_one > (offset+1))
                out=(peekb(30-(in_one-offset)/3))*4+offset;        /* X1000 */
        else
                out=offset+(peekb(30-(offset-in_one)/3))*4;        /* X1000 */
}

sin_look(in_one)   /* Provides sine of in_one from look-up-table */
unsigned int in_one;     /* in_one should be in degrees x1 with offset */
                         /* -90 to 90 degrees only */
{
        unsigned int out;
        if (in_one > (offset+1))
                out=(peekb((in_one-offset)/3))*4+offset;          /* X1000 */
        else
                out=offset-(peekb((offset-in_one)/3))*4;          /* X1000 */
}
```

```
/*****************************************************
*    John Newton  MAE 5010  Robot Project      *
*          saved under encode.c            *
*  Transforms Encoder Data into X and Y position  *
*             On the HC-11                 *
*****************************************************/
/*
*       Modified to be used in Controller Area Network (CAN)
*       by: Enrique M. Acuña, Dec. 1995
*       file name: ENCODE3.C
*/
/*****************************************************/

encode()                        /* Transforms Encoder Data into X and Y position */
{
        left_pos=left_inc;
        right_pos=right_inc;
/*      reset_count=1;
*/      left_inc=offset;
        right_inc=offset;

/* Determines direction of rotation of wheels for position data */

if (left_vel < (offset-1))
        {
                temp=left_pos;
                left_pos=mult(temp,offset-1);
        }
if (right_vel < (offset-1))
        {
                temp=right_pos;
                right_pos=mult(temp,offset-1);
        }

        temp=add(left_disp,left_pos);
        left_disp=temp;
        temp=add(right_disp,right_pos);
        right_disp=temp;


/*  theta+=(right_pos-left_pos)/L*180/pi)  */
        temp=sub(right_pos,left_pos);
        temp1=mult(temp,in_p_p);        /* inches X10 */
        temp2=mult(temp1,34+offset);      /* degrees X100 */
        temp=add(temp2,theta);
        theta=temp;

if (theta > (18000+offset+1))       /* theta should not exceed 180 degrees */
        {
                temp=theta;
                theta=temp-36000;
        }
                                    /* theta should not go below -180 degrees */
if (theta < (offset-18000-1))  /* theta should not go below -180 degrees */
```

```
        {
                temp=theta;
                theta=temp+36000;
        }

        angle=shift_left(theta,100);            /* x1 */
        ang=angle;
        if (angle < (offset-1))
                ang=add(angle,360+offset);

/* cosine and sine functions are only valid from -90 to 90 degrees */
if (angle > (90+offset+1))
        {
                temp1=shift_left(mult(sin_look(angle-90),offset-1),10);
                temp2=shift_left(cos_look(angle-90),10);
        }
else if (angle < (offset-90-1))
        {
                temp1=shift_left(sin_look(angle+90),10);
                temp2=shift_left(mult(cos_look(angle+90),offset-1),10);
        }
else
        {
                temp1=shift_left(cos_look(angle),10);           /* x100 */
                temp2=shift_left(sin_look(angle),10);           /* x100 */
        }



/*      x_inc=(right_pos+left_pos)/2*cos(theta); */
/*      temp=div(mult(add(right_pos,left_pos),10+offset),2+offset);  /* x10 */
        temp=div(mult(add(right_pos,left_pos),12+offset),2+offset);  /* x10 */
        x_inc=shift_left(mult(mult(temp,temp1),in_p_p),1000);           /* x10 */

/*      y_inc=(right_pos+left_pos)/2*sin(theta); */
        y_inc=shift_left(mult(mult(temp,temp2),in_p_p),1000);     /* x10 */


        temp=add(x_inc,x10_pos);
        x10_pos=temp;                   /* X10 */
        x_pos=shift_left(x10_pos,10);   /* X1 */

        temp=add(y_inc,y10_pos);
        y10_pos=temp;                   /* X10 */
        y_pos=shift_left(y10_pos,10);           /* X1 */

}
```

```
/*********************************************************
*     John Newton   MAE 5010   Robot Project        *
*            saved under look_tab.h             *
*          Look-up-table for sin function         *
*         prop_gain, int_gain, and u_max           *
*          are read from this look-up-table       *
*               On the HC-11                 *
*********************************************************/
/*
*          Modified by Enrique Acuña, Nov. 1995
*/
/*********************************************************/
look_tab()
{
        /* Sin of 0 degrees is 0 */
#asm
                LDAA #0
                STAA $0
#endasm
        pokeb(1,13);
        pokeb(2,26);
        pokeb(3,39);
        pokeb(4,52);
        pokeb(5,65);
        pokeb(6,77);
        pokeb(7,90);
        pokeb(8,102);
        pokeb(9,113);
        pokeb(10,125);
        pokeb(11,136);
        pokeb(12,147);
        pokeb(13,157);
        pokeb(14,167);
        pokeb(15,177);
        pokeb(16,186);
        pokeb(17,194);
        pokeb(18,202);
        pokeb(19,210);
        pokeb(20,217);
        pokeb(21,223);
        pokeb(22,228);
        pokeb(23,233);
        pokeb(24,238);
        pokeb(25,241);
        pokeb(26,245);
        pokeb(27,247);
        pokeb(28,249);
        pokeb(29,250);
        pokeb(30,250);  /* sin of 90 degrees is 1 */

        pokeb(0x20,5);          /* prop_gain */
        pokeb(0x21,3);          /* int_gain */
        pokeb(0x22,70);          /* u_max */
        pokeb(0x23,3);          /* theta_prop */
```

```
        pokeb(0x24,1);          /* theta_int */
}
```

APPENDIX D

SOFTWARE FOR SERIAL COMMUNICATIONS SECTION

AND THEGATEWAY

```
/*************************************************************
*
*       Software for the Gateway to handle the protocols for
*       the SCS and the CAN.
*       Created by: Enrique M. Acuña, Nov. 1995
*       File name: GATE1A.C
*
*************************************************************/

/* Definition of starting addresses */

#code   0x20F0
#data   0x2000
#include <startup.c>

/* Including of routines for CAN initialization */

#include "can_ini.c"

/* Register bank mapping */

#define  REG_BASE 0x1000
#include <68hc11.h>
#include <82527.h>

/* File including section */

#include <stdio.h>
#include <delay.c>
#include <SER_in.c>
#include <SER_out.c>

/* Section for global variables and constants */

unsigned char d0,d2;        /* first and third bytes of the PGN */
unsigned char d[10];        /* collects the data of the Acknowledgment message from Propulsion */
unsigned char n,l,r;        /* single ASCII character, left and right nibbles of the ASCII character
                            respectively */
unsigned char bufRX[24]; /* buffer for the reception of messages */
unsigned int res_rec;       /* response or message received flag */
unsigned int i,mess;        /* counter and message object number */
unsigned int PF_req;        /* PF number of a requested PGN*/
unsigned int des_vel;       /* desired speed from supervisor */
unsigned int des_phi;       /* desired steering angle from supervisor */
unsigned int mot_onoff;     /* emergency stop from supervisor */
unsigned int a_v;           /* average velocity or current speed from propulsion */
unsigned int angle;         /* current heading angle or compass */
unsigned int t_d;           /* current theta dot or rate of rotation */
unsigned int Xposition,Yposition;   /* current X and Y position from propulsion */
/* distances from sonars to objects from vision module */
unsigned int Son1,Son2;
unsigned int Son3,Son4,Son5,Son6;

/* Parses a character (hexadecimal number) into two nibbles of 4 bits,
```

```
        left and right nibbles, and produces the corresponding ascii
        characters for each one of them ('0' to '9' and 'A' to 'F') */
ascii(n)
unsigned char n;
{
l=((n>>4) & 0x0F) + 0x30;
r=(n & 0x0F) + 0x30;
if (l>0x3A)
        l+=0x07;
if (r>0x3A)
        r+=0x07;
}

/* takes two ascii characters ('0' to '9' and 'A' to 'F') and
        joins them into one single character (hexadecimal number
        from 0x00 to 0xFF) */
i_ascii(ln,rn)
unsigned int ln,rn;
{
if (ln>58)
        ln-=7;
if (rn>58)
        rn-=7;
n=((ln-0x30) << 4) + (rn-0x30);
return n;
}

/* Receive routine: Here the typical receive routine recommended by the
SAE J1939 for Data Link Layer is implemented partially, because only
specific requests and PDU1 format (DA specific) are used */

interrupt irq()
{
d_int();
mess = MES_BASE + (peekb(INT_REG)-3)*16;
if (peekb(INT_REG)>0x02)  {
                        pokeb(mess+CONT1,0xFD);  /* clear NewDat */
/*                      PF = (peek(mess+ARB0) >> 3) && 0x00FF;
                        DA = (peek(mess+ARB1) >> 3) && 0x00FF;
                        SA = (peek(mess+ARB2) >> 3) && 0x00FF;
                        if (PF < 240) {
*/                      switch (mess) {
                                case 0xCA10:  /* MESS1 */
                                        for(i=0;i<7;i++)
                                                d[i]=peekb(MESS1+DATA0+i);
                                        res_rec=0;
                                        break;
                                case 0xCA20:  /* MESS2 */
                                        a_v=peek(MESS2+DATA0);
                                        angle=peek(MESS2+DATA2);
                                        t_d=peek(MESS2+DATA4);
                                        res_rec=0;
                                        break;
                                case 0xCA30:  /* MESS3 */
```

```
                                        Xposition=peek(MESS3+DATA0);
                                        Yposition=peek(MESS3+DATA2);
                                        res_rec=0;
                                        break;
                                case 0xCA40:  /* MESS4 */
                                        Son1=peek(MESS4+DATA0);
                                        Son2=peek(MESS4+DATA2);
                                        Son3=peek(MESS4+DATA4);
                                        Son4=peek(MESS4+DATA6);
                                        res_rec=0;
                                        break;
                                case 0xCA50:  /* MESS5 */
                                        Son5=peek(MESS5+DATA0);
                                        Son6=peek(MESS5+DATA2);
                                        res_rec=0;
                                        break;
                                default:
                                for (i=0;i<5;i++) {
                                pokeb(PORT1,0xAA);
                                delay(1000);
                                pokeb(PORT1,0x00);
                                delay(1000); }
                                }
                        }
        pokeb(mess+CONT0,0xFD);     /* to reset IntPnd */
        e_int();
}

main()
{

/* Section of initialization of CAN */

#asm
*can development for TX and RX
*
* ADDRESS CONSTANTS
*
RESETH1 EQU $CC00
RESETLO EQU $CB00
CTLREG  EQU $CA00
CPORT1  EQU $CA9F
CPORT2  EQU $CAAF
PORT1   EQU $CADF
PORT2   EQU $CACF
CPUIFR  EQU $CA02
*
* START MAIN PROGRAM
*
* Initialize CAN
*
                LDS #$7FBF    * LOAD STACK
                SEI
                LDX $1000
```

```
                JSR INIT        * ROUTINE FOR STARTING UP OF CAN
                JSR INITC       * ROUTINE FOR INITIALIZING CAN REGISTERS
                JSR PORTS
                LDAA #$00       *CLEAR STATUS REGISTER
                STAA $CA01
                LDAA #$02       *ENABLE INTERRPUTS ON CAN
                STAA $CA00
#endasm

/* Section for setting or configuring the message objects to be used */

set_mes1();
set_mes2();
set_mes3();
set_mes4();
set_mes5();
set_mes6();
set_mes7();
set_mes9();

        bit_set(REG_BASE+TFLG2,0x80);           /* Clear the Timer Overflow Flag */
        bit_set(REG_BASE+TMSK2,0x80);           /* Enable the Timer Overflow
                                                   Interrupt*/

poke(0xF0,irq); /*IRQ via IQR pin */
e_int();
res_rec=0;
/*
* START LOOP
*/

while(1) {
while(!(SER_in()==0x09));               /* waits for start of reception */
i=0;
while(13 != (bufRX[i++]=SER_in()));     /* receives char until Carr. Ret. */
switch (bufRX[0]) {                     /* checks number of message */
        case 0x01:
                des_vel=(i_ascii(bufRX[1],bufRX[2]) << 8) + i_ascii(bufRX[3],bufRX[4]);
                des_phi=(i_ascii(bufRX[5],bufRX[6]) << 8) + i_ascii(bufRX[7],bufRX[8]);
                mot_onoff=(i_ascii(bufRX[9],bufRX[10]) << 8) + i_ascii(bufRX[11],bufRX[12]);
                upd6();
                send(MESS6);
                while(res_rec==1);
                SER_out(0x09);
                SER_out(0x01);
                for(i=0;i<7;i++) {
                        ascii(d[i]);
                        SER_out(l);
                        SER_out(r);
                        }
                break;
        case 0x02:
                d0=i_ascii(bufRX[1],bufRX[2]);
                PF_req=i_ascii(bufRX[3],bufRX[4]);
```

```
            d2=i_ascii(bufRX[5],bufRX[6]);
            upd7();
            send(MESS7);
            while(res_rec==1);
            SER_out(0x09);
            SER_out(0x02);

            ascii(a_v>>8);
            SER_out(l);
            SER_out(r);
            ascii(a_v);
            SER_out(l);
            SER_out(r);

            ascii(angle>>8);
            SER_out(l);
            SER_out(r);
            ascii(angle);
            SER_out(l);
            SER_out(r);

            ascii(t_d>>8);
            SER_out(l);
            SER_out(r);
            ascii(t_d);
            SER_out(l);
            SER_out(r);
            break;
case 0x03:
            d0=i_ascii(bufRX[1],bufRX[2]);
            PF_req=i_ascii(bufRX[3],bufRX[4]);
            d2=i_ascii(bufRX[5],bufRX[6]);
            upd7();
            send(MESS7);
            while(res_rec==1);
            SER_out(0x09);
            SER_out(0x03);

            ascii(Xposition>>8);
            SER_out(l);
            SER_out(r);
            ascii(Xposition);
            SER_out(l);
            SER_out(r);

            ascii(Yposition>>8);
            SER_out(l);
            SER_out(r);
            ascii(Yposition);
            SER_out(l);
            SER_out(r);
            break;
case 0x04:
            d0=i_ascii(bufRX[1],bufRX[2]);
```

```
        PF_req=i_ascii(bufRX[3],bufRX[4]);
        d2=i_ascii(bufRX[5],bufRX[6]);
        upd9();
        send(MESS9);
        while(res_rec==1);
        SER_out(0x09);
        SER_out(0x04);

        ascii(Son1>>8);
        SER_out(l);
        SER_out(r);
        ascii(Son1);
        SER_out(l);
        SER_out(r);

        ascii(Son2>>8);
        SER_out(l);
        SER_out(r);
        ascii(Son2);
        SER_out(l);
        SER_out(r);

        ascii(Son3>>8);
        SER_out(l);
        SER_out(r);
        ascii(Son3);
        SER_out(l);
        SER_out(r);

        ascii(Son4>>8);
        SER_out(l);
        SER_out(r);
        ascii(Son4);
        SER_out(l);
        SER_out(r);
        break;
case 0x05:
        d0=i_ascii(bufRX[1],bufRX[2]);
        PF_req=i_ascii(bufRX[3],bufRX[4]);
        d2=i_ascii(bufRX[5],bufRX[6]);
        upd9();
        send(MESS9);
        while(res_rec==1);
        SER_out(0x09);
        SER_out(0x05);

        ascii(Son5>>8);
        SER_out(l);
        SER_out(r);
        ascii(Son5);
        SER_out(l);
        SER_out(r);

        ascii(Son6>>8);
```

```
                SER_out(l);
                SER_out(r);
                ascii(Son6);
                SER_out(l);
                SER_out(r);
                break;
        default:
                for(i=0;i<5;i++) {
                        pokeb(PORT1,0xAA);
                        delay(1000);
                        pokeb(PORT1,0x00);
                        delay(1000);
                        }
                break;
        }
        }
}

/* set message objetct #1 for receiving ACK of command (with desired
        speed and steering angle) from Navig. */

set_mes1() {
        pokeb(MESS1+CONT0,0x55);
        pokeb(MESS1+CONT1,0x55);
        pokeb(MESS1+ARB0,0xC7);
        pokeb(MESS1+ARB1,0x40);
        pokeb(MESS1+ARB2,0x00);
        pokeb(MESS1+ARB3,0x08);
        pokeb(MESS1+MCONF,0x84);
        pokeb(MESS1+CONT0,0x99);
}

/* set message object #2 for receiving data (RS_C_td) from Navig. */

set_mes2() {
        pokeb(MESS2+CONT0,0x55);
        pokeb(MESS2+CONT1,0x55);
        pokeb(MESS2+ARB0,0xC7);
        pokeb(MESS2+ARB1,0x68);
        pokeb(MESS2+ARB2,0x00);
        pokeb(MESS2+ARB3,0x08);
        pokeb(MESS2+MCONF,0x84);
        pokeb(MESS2+CONT0,0x99);
}

/* set message objetct #3 for receiving data (XYposition) from Nav. */

set_mes3() {
        pokeb(MESS3+CONT0,0x55);
        pokeb(MESS3+CONT1,0x55);
        pokeb(MESS3+ARB0,0xC7);
        pokeb(MESS3+ARB1,0x60);
        pokeb(MESS3+ARB2,0x00);
        pokeb(MESS3+ARB3,0x08);
```

```
        pokeb(MESS3+MCONF,0x84);
        pokeb(MESS3+CONT0,0x99);
}
/* set message objetct #4 for receiving response with Son1-4 from Vision */

set_mes4() {
        pokeb(MESS4+CONT0,0x55);
        pokeb(MESS4+CONT1,0x55);
        pokeb(MESS4+ARB0,0xC7);
        pokeb(MESS4+ARB1,0x58);
        pokeb(MESS4+ARB2,0x00);
        pokeb(MESS4+ARB3,0x10);
        pokeb(MESS4+MCONF,0x84);
        pokeb(MESS4+CONT0,0x99);
}
/* set message objetct #5 for receiving response with Son5-6 from Vision */

set_mes5() {
        pokeb(MESS5+CONT0,0x55);
        pokeb(MESS5+CONT1,0x55);
        pokeb(MESS5+ARB0,0xC7);
        pokeb(MESS5+ARB1,0x48);
        pokeb(MESS5+ARB2,0x00);
        pokeb(MESS5+ARB3,0x10);
        pokeb(MESS5+MCONF,0x84);
        pokeb(MESS5+CONT0,0x99);
}

/* set message objetct #6 for transmiting command with desired speed and
        steering angle to Navigation */

set_mes6() {
        pokeb(MESS6+CONT0,0x55);
        pokeb(MESS6+CONT1,0x59);
        pokeb(MESS6+ARB0,0xC7);
        pokeb(MESS6+ARB1,0x70);
        pokeb(MESS6+ARB2,0x08);
        pokeb(MESS6+ARB3,0x00);
        pokeb(MESS6+MCONF,0x8C);
        pokeb(MESS6+CONT0,0xBF);
}

/* set message objetct #7 for transmiting request of RS_C_td and
        XYposition to Naviagation */

set_mes7() {
        pokeb(MESS7+CONT0,0x55);
        pokeb(MESS7+CONT1,0x59);
        pokeb(MESS7+ARB0,0xC7);
        pokeb(MESS7+ARB1,0x50);
        pokeb(MESS7+ARB2,0x08);
        pokeb(MESS7+ARB3,0x00);
        pokeb(MESS7+MCONF,0x8C);
        pokeb(MESS7+CONT0,0xBF);
```

```
}

/* set message objetct #9 for transmiting request of Son1-4
        and Son5-6 to Vision */

set_mes9() {
        pokeb(MESS9+CONT0,0x55);
        pokeb(MESS9+CONT1,0x59);
        pokeb(MESS9+ARB0,0xC7);
        pokeb(MESS9+ARB1,0x50);
        pokeb(MESS9+ARB2,0x10);
        pokeb(MESS9+ARB3,0x00);
        pokeb(MESS9+MCONF,0x8C);
        pokeb(MESS9+CONT0,0xBF);
}


/* ////////////////
* Update data in message object 6
*/
upd6() {

        pokeb(MESS6+CONT1,0xFA);  /*START OF UPDATING (CPUUpd=NewDat=1)*/

        poke(MESS6+DATA0,des_vel); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        poke(MESS6+DATA2,des_phi);
        poke(MESS6+DATA4,mot_onoff);
/*      pokeb(MESS6+DATA6,start);
*/
        pokeb(MESS6+CONT1,0xF7);   /*END OF DATA UPDATING (CPUUpd=0)*/
 }
/* ////////////////
* Update data in message object 7
*/
upd7() {

        pokeb(MESS7+CONT1,0xFA);  /*START OF UPDATING (CPUUpd=NewDat=1)*/

        pokeb(MESS7+DATA0,d0); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        pokeb(MESS7+DATA1,PF_req);
        pokeb(MESS7+DATA2,d2);

        pokeb(MESS7+CONT1,0xF7);   /*END OF DATA UPDATING (CPUUpd=0)*/
 }
/* ////////////////
* Update data in message object 9
*/
upd9() {

        pokeb(MESS9+CONT1,0xFA);  /*START OF UPDATING (CPUUpd=NewDat=1)*/

        pokeb(MESS9+DATA0,d0); /*UPDATE DATA BYTES (SEND DATA BYTE1)*/
        pokeb(MESS9+DATA1,PF_req);
        pokeb(MESS9+DATA2,d2);
```

```
        pokeb(MESS9+CONT1,0xF7);    /*END OF DATA UPDATING (CPUUpd=0)*/
 }
/*
* MESSAGE OBJECT SENDS ITS DATA FRAME
*/
send(mess)
unsigned int mess;
{
/*TXRQUST=1 CPPUUD=0 NEWDAT=1*/
pokeb(mess+CONT1,0xE6);
res_rec=1;
}
```

```
/*************************************************************************
*
*       Primary Supervisor's Software for testing SCS and gateway functions
*       Created by: Enrique M. Acuña, Nov. 1995
*       File name: PRIM_SUP.C
*
*************************************************************************/

#include <math.h>
#include <conio.h>
#include "ibmcom3.c"     /* routines for low-level serial communications on the IBM PC */

/* PF numbers for the messages to be stablished */

#define RS_C_Cd_PF 0xED         /* 237 */
#define XY_pos_PF  0xEC         /* 236 */
#define Son1to4_PF 0xEB         /* 235 */
#define Son56_PF   0xE9 /* 233 */


#define H       18.88   /* length from rear axis of the wheelchair to the front one */
#define R       8.06    /* radius of rear wheels of the wheelchair */

extern int time_out;            /* variable defined in IBMCOM3.C to set a time to account for a time
                                out*/

char n,l,r;             /* single ASCII character, left and right nibble that form the single ASCII
                        character */

int RoadSpeed   ;       /* Desired velocity to be sent from supervisor to propulsion */
int SteeringAngle;      /* Desired steering angle to be sent from supervisor to propulsion */
int Range[7];           /* Six sonars' readings from vision */
int choice;             /* action or message (number) to send */
int OFF;                /* status of the emergency stop */

float ARoadSpeed,Compass,Comp_dot;      /* current speed, heading angle and rate of rotation */
float Xposition,Yposition;              /* current XY position */
double ASteeringAngle;                  /* current steering angle */

/* to form the screen with the menu to be display */
void title(void);
void action1(void);
void choose_action(void);

void ascii(unsigned char);              /* see at the end */
char i_ascii(unsigned char, unsigned char);  /* see at the end */

void main(void)
{
        #define SPEED 9600
        unsigned char bufTX[24];
        unsigned char bufRX[24];
        unsigned char buf[24];
        unsigned int temp;
```

```
char i,j;
init_port(2, SPEED,COM_NONE,1);
clrscr();
title();
bufTX[0]=0x09; /* 0x09 (buf[0]) starts mess to TX */
do {
        CLEAR_BUFFERS;
choose_action();
switch(choice) {
        case 1:
                action1();
                bufTX[1]=0x01; /* indicates message #1 */
                ascii(RoadSpeed>>8);
                bufTX[2]=l;
                bufTX[3]=r;
                ascii(RoadSpeed);
                bufTX[4]=l;
                bufTX[5]=r;
                ascii(SteeringAngle>>8);
                bufTX[6]=l;
                bufTX[7]=r;
                ascii(SteeringAngle);
                bufTX[8]=l;
                bufTX[9]=r;
                ascii(OFF>>8);
                bufTX[10]=l;
                bufTX[11]=r;
                ascii(OFF);
                bufTX[12]=l;
                bufTX[13]=r;
                send_command(bufTX);
                wait_for_response(bufRX);

                if (time_out < 1)
                        break;

                buf[0]=i_ascii(bufRX[2],bufRX[3]);
                buf[1]=i_ascii(bufRX[4],bufRX[5]);
                buf[2]=i_ascii(bufRX[6],bufRX[7]);
                buf[3]=i_ascii(bufRX[8],bufRX[9]);
                buf[4]=i_ascii(bufRX[10],bufRX[11]);
                buf[5]=i_ascii(bufRX[12],bufRX[13]);
                buf[6]=i_ascii(bufRX[14],bufRX[15]);
                buf[7]=i_ascii(bufRX[16],bufRX[17]);
                if (bufRX[1]==0x01) { /* checks if correct mess is received */
                        gotoxy(1,16);
                        printf("Acknowledged: %x %x %x %x %x %x %x
                        %x",buf[0],buf[1],buf[2],buf[3],buf[4],buf[5],buf[6],buf[7]);
                        break;
                        }
                else {
                        gotoxy(1,16);
                        printf("Not_Acknowledged");
                        break;
```

```
                }
case 2:
        bufTX[1]=0x02;  /* indicates message #2 */
        bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
        bufTX[3]=0x30;
        bufTX[4]=0x45;  /* bufTX[4] and [5] send RS_C_Cd_PF (0xED) */
        bufTX[5]=0x44;  /* in ascii */
        bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
        bufTX[7]=0x30;
        gotoxy(1,18);
        printf("RS_C_Cd requested   ");
        send_command(bufTX);
        wait_for_response(bufRX);

        if (time_out < 1)
                break;

        buf[0]=i_ascii(bufRX[2],bufRX[3]);
        buf[1]=i_ascii(bufRX[4],bufRX[5]);
        buf[2]=i_ascii(bufRX[6],bufRX[7]);
        buf[3]=i_ascii(bufRX[8],bufRX[9]);
        buf[4]=i_ascii(bufRX[10],bufRX[11]);
        buf[5]=i_ascii(bufRX[12],bufRX[13]);
        if (bufRX[1]==0x02) {
                temp=(buf[0]<<8) + buf[1];
                ARoadSpeed=(temp-32767.0)/10;

/*              printf("buf[0:1] %x %x buf[2,3] %x %x
                ",buf[0],buf[1],buf[2],buf[3]);
                printf("buf[4,5] %x %x ",buf[4],buf[5]);
*/              temp=(buf[2]<<8) + buf[3];
                Compass=temp-32767.0;
                temp=(buf[4]<<8) + buf[5];
                Comp_dot=(temp-32767.0)/10;
//              ASteeringAngle=atan(Comp_dot*H/(ARoadSpeed*R));
                gotoxy(1,19);
                printf("Road Speed: %5.2f",ARoadSpeed);
                gotoxy(1,20);
                printf("Compass:   %5.2f",Compass);
                gotoxy(1,21);
                printf("AS.Angle:   %lf ",Comp_dot);
                break;
                }
        else {
                gotoxy(1,19);
                printf("No_expected_response");
                break;
                }
case 3:
        bufTX[1]=0x03;  /* indicates message #3 */
        bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
        bufTX[3]=0x30;
        bufTX[4]=0x45;  /* bufTX[4] and [5] send XY_pos_PF (0xEC) */
        bufTX[5]=0x43;  /* in ascii */
```

```
        bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
        bufTX[7]=0x30;
        gotoxy(1,23);
        printf("XY_pos requested   ");
        send_command(bufTX);
        wait_for_response(bufRX);

        if (time_out < 1)
                break;

        buf[0]=i_ascii(bufRX[2],bufRX[3]);
        buf[1]=i_ascii(bufRX[4],bufRX[5]);
        buf[2]=i_ascii(bufRX[6],bufRX[7]);
        buf[3]=i_ascii(bufRX[8],bufRX[9]);
        if (bufRX[1]==0x03) {
                Xposition=((buf[0]<<8) + buf[1])-32767;
                Yposition=((buf[2]<<8) + buf[3])-32767;
                gotoxy(1,24);
                printf("X-Y position %5.2f %5.2f ",Xposition,Yposition);
                break;
                }
        else {
                gotoxy(1,24);
                printf("No_expected_response");
                break;
                }
case 4:
        bufTX[1]=0x04;  /* indicates message #4 */
        bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
        bufTX[3]=0x30;
        bufTX[4]=0x45;  /* bufTX[4] and [5] send Son1to4_PF (0xEB) */
        bufTX[5]=0x42;  /* in ascii */
        bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
        bufTX[7]=0x30;
        gotoxy(35,13);
        printf("Son1 to 4 requested   ");
        send_command(bufTX);
        wait_for_response(bufRX);

        if (time_out < 1)
                break;

        buf[0]=i_ascii(bufRX[2],bufRX[3]);
        buf[1]=i_ascii(bufRX[4],bufRX[5]);
        buf[2]=i_ascii(bufRX[6],bufRX[7]);
        buf[3]=i_ascii(bufRX[8],bufRX[9]);
        buf[4]=i_ascii(bufRX[10],bufRX[11]);
        buf[5]=i_ascii(bufRX[12],bufRX[13]);
        buf[6]=i_ascii(bufRX[14],bufRX[15]);
        buf[7]=i_ascii(bufRX[16],bufRX[17]);
        if (bufRX[1]==0x04) {
                Range[0]=(buf[0]<<8) + buf[1];
                Range[1]=(buf[2]<<8) + buf[3];
                Range[2]=(buf[4]<<8) + buf[5];
```

```
                        Range[3]=(buf[6]<<8) + buf[7];
                        gotoxy(35,14);
                        printf("Sonar 1-2: %d  %d  ",Range[0],Range[1]);
                        gotoxy(35,15);
                        printf("Sonar 3-4: %d  %d  ",Range[2],Range[3]);
                        break;
                        }
                else {
                        gotoxy(35,14);
                        printf("No_expected_response");
                        break;
                        }
        case 5:
                bufTX[1]=0x05;  /* indicates message #5 */
                bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
                bufTX[3]=0x30;
                bufTX[4]=0x45;  /* bufTX[4] and [5] send Son56_PF (0xE9) */
                bufTX[5]=0x39;  /* in ascii */
                bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
                bufTX[7]=0x30;
                gotoxy(35,17);
                printf("Sonar 5-6 requested   ");
                send_command(bufTX);
                wait_for_response(bufRX);

                if (time_out < 1)
                        break;

                buf[0]=i_ascii(bufRX[2],bufRX[3]);
                buf[1]=i_ascii(bufRX[4],bufRX[5]);
                buf[2]=i_ascii(bufRX[6],bufRX[7]);
                buf[3]=i_ascii(bufRX[8],bufRX[9]);
                if (bufRX[1]==0x05) {
                        Range[4]=(buf[0]<<8) + buf[1];
                        Range[5]=(buf[2]<<8) + buf[3];
                        gotoxy(35,18);
                        printf("Sonar 5-6: %d  %d  ",Range[4],Range[5]);
                        break;
                        }
                else {
                        gotoxy(35,18);
                        printf("No_expected_response");
                        break;
                        }
        case 6:
                exit(1);
                break;
        default:
                gotoxy(1,19);
                printf("Error");
                break;
        }
} while(!(choice==6));
clrscr();
```

```
}

void title(void)
{
        gotoxy(25,1);
        printf("Test of Autonomous Vehicle");
        gotoxy(25,2);
        printf("PC-ENAT-CAN communications");
        gotoxy(25,4);
        printf("1) Command D.Speed, D.S.Angle");
        gotoxy(25,5);
        printf("2) Request ARSpeed, Compass, Comp_d");
        gotoxy(25,6);
        printf("3) Request XY_pos");
        gotoxy(25,7);
        printf("4) Request Sonar1 to 4");
        gotoxy(25,8);
        printf("5) Request Sonar5 6");
        gotoxy(25,9);
        printf("6) Exit Program");

}
void choose_action(void)
{
        int s;
        gotoxy(25,11);
        printf("Action: ");
        do {
                gotoxy(35,11);
                s=scanf("%d",&choice);
                flushall();
        } while(s==0);
        gotoxy(35,11);
        printf("%d    ",choice);
}
void action1(void)
{
        int s;
        gotoxy(1,13);
        printf("D.Speed  -42/42");
        do {
                gotoxy(20,13);
                s=scanf("%d",&RoadSpeed);
                flushall();
        } while((s==0) || (RoadSpeed>42) || (RoadSpeed<-42));
        gotoxy(20,13);
        printf("%d    ",RoadSpeed);
        RoadSpeed+=32767;

        gotoxy(1,14);
        printf("D.S.Angle -90/90");
        do {
                gotoxy(20,14);
                s=scanf("%d",&SteeringAngle);
```

```
                flushall();
        } while((s==0) || (SteeringAngle>90) || (SteeringAngle<-90));
        gotoxy(20,14);
        printf("%d    ",SteeringAngle);
        SteeringAngle+=32767;

        gotoxy(1,15);
        printf("OFF 0/1");
        do {
                gotoxy(20,15);
                s=scanf("%d",&OFF);
                flushall();
        } while((s==0) || !((OFF==0) || (OFF==1)));
        gotoxy(20,15);
        printf("%d  ",OFF);
}

/* Parses a character (hexadecimal number) into two nibbles of 4 bits,
        left and right nibbles, and produces the corresponding ascii
        characters for each one of them ('0' to '9' and 'A' to 'F') */

void ascii(n)
unsigned char n;
{
l=((n>>4) & 0x0F) + 0x30;
r=(n & 0x0F) + 0x30;
if (l>0x39)
        l+=0x07;
if (r>0x39)
        r+=0x07;
}

/* takes two ascii characters ('0' to '9' and 'A' to 'F') and
        joins them into one single character (hexadecimal number
        from 0x00 to 0xFF) */

char i_ascii(ln,rn)
unsigned char ln,rn;
{
if (ln>0x39)
        ln-=7;
if (rn>0x39)
        rn-=7;
n=((ln-0x30) << 4) + (rn-0x30);
return n;
}
```

```
/*****************************************************************************
 * DESCRIPTION:        This file contains a set of routines for doing low-level  *
 *                  serial communications on the IBM PC.  It was translated      *
 *                  directly from Wayne Conrad's IBMCOM.PAS version 3.1, with    *
 *                  the goal of near-perfect functional correspondence between   *
 *                  the Pascal and C versions.                                   *
 *                                                                               *
 * REVISIONS:  18 OCT 89 - RAC - Original translation from IBMCOM.PAS, with      *
 *                               liberal plagiarism of comments from the         *
 *                               Pascal.                                         *
 *****************************************************************************/
/*****************************************************************************
 *                                  ibmcom.c                                     *
 * last mod 12-10-95 by Enrique M. Acuña
* static void strip_cr_linefeed(char *s)
* int send_command(char *command)
* int init_port(int port, int speed,int parity, int stop)
* void wait_for_response(char *b)
*
*****************************************************************************
*/

//ibmcom.c 2/5/93
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>
#include <string.h>
#include "ibmcom.h"
#define CLEAR_SCREEN clrscr();


#define CLEAR_BUFFERS       com_flush_tx(); com_flush_rx()
#define TIME_OUT 120

int time_out;
/*****************************************************************************
 *                                  Global Data                                  *
 *****************************************************************************/
static int          com_carrier(void);
static void                 com_deinstall(void);
static void                 com_flush_rx(void);
static void                 com_flush_tx(void);
static int          com_install(int portnum);
static void interrupt       com_interrupt_driver();
static void                 com_lower_dtr(void);
static void                 com_raise_dtr(void);
static char                 com_rx(void);
static int          com_rx_empty(void);
static void                 com_set_parity(enum par_code parity, int stop_bits);
static void                 com_set_speed(unsigned speed);
static void                 com_tx(char c);
static int          com_tx_empty(void);
static int          com_tx_ready(void);
static void                 com_tx_string(char *s);
```

```
// interface to external
int                             send_command(char *command);
int                             init_port(int port, int speed,int parity, int stop);
void                            terminal();
void                            wait_for_response(char *b);

/* UART i/o addresses.  Values depend upon which COMM port is selected */

static int uart_data;                   /* Data register */
static int uart_ier;            /* Interrupt enable register */
static int uart_iir;            /* Interrupt identification register */
static int uart_lcr;            /* Line control register */
static int uart_mcr;                    /* Modem control register */
static int uart_lsr;            /* Line status register */
static int uart_msr;                    /* Modem status register */

static char     com_installed;          /* Flag: Communications routines installed */
static int intnum;                      /* Interrupt vector number for chosen port */
static char     i8259bit;               /* 8259 bit mask */
static char     old_i8259_mask;         /* Copy as it was when we were called */
static char     old_ier;                /* Modem register contents saved for */
static char     old_mcr;                /*  restoring when we're done */
static void interrupt (*old_vector)();  /* Place to save COM1 vector */

/* Transmit queue.  Characters to be transmitted are held here until the */
/* UART is ready to transmit them. */

#define TX_QUEUE_SIZE           81 //256 /* Transmit queue size.  Change to suit */

static char     tx_queue[TX_QUEUE_SIZE];
static int tx_in;                       /* Index of where to store next character */
static int tx_out;                      /* Index of where to retrieve next character */
static int tx_chars;                    /* Count of characters in queue */

/* Receive queue.  Received characters are held here until retrieved by */
/* com_rx() */

#define RX_QUEUE_SIZE 256 // 4096        /* Receive queue size.  Change to suit */

static char     rx_queue[RX_QUEUE_SIZE];
static int rx_in;                       /* Index of where to store next character */
static int rx_out;                      /* Index of where to retrieve next character */
static int rx_chars;                    /* Count of characters in queue */

/*****************************************************************************
 *                                com_install()                              *
 *****************************************************************************
 * DESCRIPTION:     Installs the communications drivers.                     *
 *                                                                           *
 * SYNOPSIS:    status = com_install(int portnum);                           *
 *              int     portnum;        Desired port number                  *
 *              int     status;         0 = Successful installation          *
 *                                      1 = Invalid port number              *
 *                                      2 = No UART for specified port       *
```

```
*                                   3 = Drivers already installed          *
*                                                                          *
* REVISIONS:  18 OCT 89 - RAC - Translated from IBMCOM.PAS                 *
*****************************************************************************/

static const int    uart_base[] =     { 0x3F8, 0x2F8, 0x3E8, 0x2E8 };
static const char   intnums[] =       { 0x0C,  0x0B,  0x0C,  0x0B };
static const char   i8259levels[] =   { 4,     3,     4,     3 };

static int com_install(int portnum) {

        if (com_installed)                              /* Drivers already installed */
        return 3;
        if ((portnum < 1) || (portnum > MAX_PORT))      /* Port number out of bounds */
        return 1;

        uart_data = uart_base[portnum-1];       /* Set UART I/O addresses */
        uart_ier  = uart_data + IER;            /*  for the selected comm */
        uart_iir  = uart_data + IIR;            /*  port */
        uart_lcr  = uart_data + LCR;
        uart_mcr  = uart_data + MCR;
        uart_lsr  = uart_data + LSR;
        uart_msr  = uart_data + MSR;
        intnum    = intnums[portnum-1];         /* Ditto for interrupt */
        i8259bit  = 1 << i8259levels[portnum-1];/*  vector and 8259 bit mask */

        old_ier = inportb(uart_ier);            /* Return an error if we */
        outportb(uart_ier, 0);                  /*  can't access the UART */
        if (inportb(uart_ier) != 0)
        return 2;

        disable();                                      /* Save the original 8259 */
        old_i8259_mask = inportb(0x21);         /*  mask, then disable the */
        outportb(0x21, old_i8259_mask | i8259bit);/*  8259 for this interrupt */
        enable();

        com_flush_tx();                         /* Clear the transmit and */
        com_flush_rx();                         /*  receive queues */

        old_vector = getvect(intnum);           /* Save old COMM vector, */
        setvect(intnum, &com_interrupt_driver); /*  then install a new one, */
        com_installed = TRUE;                   /*  and note that we did */

        outportb(uart_lcr, DATA8 + NOPAR + STOP1);      /* 8 data, no parity, 1 stop */

        disable();                                      /* Save MCR, then enable */
        old_mcr = inportb(uart_mcr);            /*  interrupts onto the bus, */
        outportb(uart_mcr,                              /*  activate RTS and leave */
                (old_mcr & DTR) | (OUT2 + RTS));        /*  DTR the way it was */
        enable();

        outportb(uart_ier, DR);                 /* Enable receive interrupts */

        disable();                                      /* Now enable the 8259 for */
```

```
        outportb(0x21, inportb(0x21) & ~i8259bit);/*  this interrupt */
        enable();
        return 0;                                       /* Successful installation */
        }                                               /* End com_install() */

/*************************************************************************
*                                com_install()                           *
*************************************************************************
* DESCRIPTION:      Denstalls the communications drivers completely, without *
*                   changing the baud rate or DTR.  It tries to leave the    *
*                   interrupt vectors and enables and everything else as they *
*                   were when the driver was installed.                      *
*                                                                            *
* NOTE:             This function MUST be called before returning to DOS, so the *
*                   interrupt vector won't point to our driver anymore, since it *
*                   will surely get overwritten by some other transient program *
*                   eventually.                                              *
*                                                                            *
* REVISIONS:  18 OCT 89 - RAC - Translated from IBMCOM.PAS                   *
*************************************************************************/

static void com_deinstall(void) {

        if (com_installed) {                    /* Don't de-install twice! */
        outportb(uart_mcr, old_mcr);            /* Restore the UART */
        outportb(uart_ier, old_ier);            /*  registers ... */
        disable();
        outportb(0x21,                          /*  ... the 8259 interrupt */
                (inportb(0x21)  & ~i8259bit) | /*  mask ... */
                (old_i8259_mask &  i8259bit));
        enable();
        setvect(intnum, old_vector);            /*  ... and the comm */
        com_installed = FALSE;                  /*  interrupt vector */
        }                                       /* End com_installed */
        }                                               /* End com_deinstall() */

/*************************************************************************
*                                com_set_speed()                         *
*************************************************************************
* DESCRIPTION:      Sets the baud rate.                                      *
*                                                                            *
* SYNOPSIS:   void com_set_speed(unsigned speed);                            *
*             unsigned speed;                 Desired baud rate              *
*                                                                            *
* NOTES:      The input parameter can be anything between 2 and 65535.       *
*             However, I (Wayne) am not sure that extremely high speeds      *
*             (those above 19200) will always work, since the baud rate      *
*             divisor will be six or less, where a difference of one can     *
*             represent a difference in baud rate of 3840 bits per second    *
*             or more.)                                                      *
*                                                                            *
* REVISIONS:  18 OCT 89 - RAC - Translated from IBMCOM.PAS                   *
*************************************************************************/
```

```
static void com_set_speed(unsigned speed) {

        unsigned        divisor;                /* A local temp */

        if (com_installed) {
        if (speed < 2) speed = 2;               /* Force proper input */
        divisor = 115200L / speed;                      /* Recond baud rate divisor */
        disable();                                      /* Interrupts off */
        outportb(uart_lcr,                      /* Set up to load baud rate */
                inportb(uart_lcr) | 0x80); /*  divisor into UART */
        outport(uart_data, divisor);                    /* Do so */
        outportb(uart_lcr,                      /* Back to normal UART ops */
                inportb(uart_lcr) & ~0x80);
        enable();                               /* Interrupts back on */
        }                                               /* End "comm installed" */
        }                                                       /* End com_set_speed() */

/*****************************************************************************
 *                              com_set_parity()                             *
 *****************************************************************************
 * DESCRIPTION: Sets the parity and stop bits.                               *
 *                                                                           *
 * SYNOPSIS:    void com_set_parity(enum par_code parity, int stop_bits);    *
 *              int     code;           COM_NONE = 8 data bits, no parity    *
 *                                      COM_EVEN = 7 data, even parity       *
 *                                      COM_ODD  = 7 data, odd parity        *
 *                                      COM_ZERO = 7 data, parity bit = zero *
 *                                      COM_ONE  = 7 data, parity bit = one  *
 *              int     stop_bits;      Must be 1 or 2                       *
 *                                                                           *
 * REVISIONS:   18 OCT 89 - RAC - Translated from the Pascal                 *
 *****************************************************************************/

static const char  lcr_vals[] = {
                        DATA8 + NOPAR,
                        DATA7 + EVNPAR,
                        DATA7 + ODDPAR,
                        DATA7 + STKPAR,
                        DATA7 + ZROPAR
                        } ;

static void com_set_parity(enum par_code parity, int stop_bits) {
        disable();
        outportb(uart_lcr, lcr_vals[parity] | ((stop_bits == 2) ? STOP2 : STOP1));
        enable();
        }                                                       /* End com_set_parity() */

/*****************************************************************************
 *                              com_raise_dtr()                              *
 *                              com_lower_dtr()                              *
 *****************************************************************************
 * DESCRIPTION:         These routines raise and lower the DTR line.  Lowering DTR  *
 *              causes most modems to hang up.                               *
 *                                                                           *
```

```
* REVISIONS:  18 OCT 89 - RAC - Translated from the Pascal.                *
*****************************************************************************/

static void com_lower_dtr(void) {
        if (com_installed) {
                disable();
        outportb(uart_mcr, inportb(uart_mcr) & ~DTR);
                enable();
        }                                       /* End 'comm installed' */
        }                                               /* End com_raise_dtr() */

static void com_raise_dtr(void) {
        if (com_installed) {
                disable();
        outportb(uart_mcr, inportb(uart_mcr) | DTR);
                enable();
        }                                       /* End 'comm installed' */
        }                                               /* End com_lower_dtr() */

/*****************************************************************************
*                                  com_tx()                                  *
*                               com_tx_string()                              *
*****************************************************************************
* DESCRIPTION: Transmit routines.  com_tx() sends a single character by     *
*              waiting until the transmit buffer isn't full, then putting   *
*              the character into it.  The interrupt driver will then send  *
*              the character once it is at the head of the transmit queue   *
*              and a transmit interrupt occurs.  com_tx_string() sends a    *
*              string by repeatedly calling com_tx().                       *
*                                                                           *
* SYNOPSES:  void    com_tx(char c);          Send the character c          *
*            void    com_tx_string(char *s);  Send the string s            *
*                                                                           *
* REVISIONS:  18 OCT 89 - RAC - Translated from the Pascal                  *
*****************************************************************************/

static void com_tx(char c) {
        if (com_installed) {
        while (!com_tx_ready()) ;               /* Wait for non-full buffer */
        disable();                              /* Interrupts off */
        tx_queue[tx_in++] = c;                  /* Stuff character in queue */
        if (tx_in == TX_QUEUE_SIZE) tx_in = 0;  /* Wrap index if needed */
        tx_chars++;                             /* Number of char's in queue */
        outportb(uart_ier,                      /* Enable UART tx interrupt */
                inportb(uart_ier) | THRE);
        enable();                               /* Interrupts back on */
        }                                       /* End 'comm installed' */
        }                                               /* End com_tx() */

static void com_tx_string(char *s) {
        while (*s)
                {
//              delay(5);
                delay(3);
```

```
                com_tx(*s++);                   /* Send the string! */
                }
        }                                               /* End com_tx_string() */

/*****************************************************************************
*                                  com_rx()                                  *
*****************************************************************************
* DESCRIPTION:          Returns the next character from the receive buffer, or a  *
*               NULL character ('\0') if the buffer is empty.           *
*                                                                               *
* SYNOPSIS:     c = com_rx();                                                   *
*               char    c;                      The returned character      *
*                                                                               *
* REVISIONS:    18 OCT 89 - RAC - Translated from the Pascal.                   *
*****************************************************************************/

static char         com_rx(void) {

        char    rv;                             /* Local temp */

        if (!rx_chars || !com_installed)        /* Return NULL if receive */
        return '\0';                    /*  buffer is empty */
        disable();                              /* Interrupts off */
        rv = rx_queue[rx_out++];                /* Grab char from queue */
        if (rx_out == RX_QUEUE_SIZE)            /* Wrap index if needed */
        rx_out = 0;
        rx_chars--;                             /* One less char in queue */
        enable();                               /* Interrupts back on */
        return rv;                              /* The answer! */
        }                                       /* End com_rx() */

/*****************************************************************************
*                           Queue Status Routines                           *
*****************************************************************************
* DESCRIPTION:          Small routines to return status of the transmit and receive  *
*               queues.                                                         *
*                                                                               *
* REVISIONS:    18 OCT 89 - RAC - Translated from the Pascal.                   *
*****************************************************************************/

static int com_tx_ready(void) {                 /* Return TRUE if the */
        return ((tx_chars < TX_QUEUE_SIZE) ||   /*  transmit queue can */
                (!com_installed));                      /*  accept a character */
        }                                               /* End com_tx_ready() */

static int com_tx_empty(void) {                 /* Return TRUE if the */
        return (!tx_chars || (!com_installed));         /*  transmit queue is empty */
        }                                               /* End com_tx_empty() */

static int com_rx_empty(void) {                 /* Return TRUE if the */
        return (!rx_chars || (!com_installed));         /*  receive queue is empty */
        }                                               /* End com_tx_empty() */

/*****************************************************************************
```

```
*                                com_flush_tx()                                *
*                                com_flush_rx()                                *
*******************************************************************************
* DESCRIPTION:        Buffer flushers!  These guys just initialize the transmit *
*               and receive queues (respectively) to their empty state.        *
*                                                                               *
* REVISIONS:  18 OCT 89 - RAC - Translated from the Pascal                      *
*******************************************************************************/

static void com_flush_tx() { disable(); tx_chars = tx_in = tx_out = 0; enable(); }
void com_flush_rx() { disable(); rx_chars = rx_in = rx_out = 0; enable(); }

/*******************************************************************************
*                                com_carrier()                                 *
*******************************************************************************
* DESCRIPTION:        Returns TRUE if a carrier is present.                     *
*                                                                               *
* REVISIONS:  18 OCT 89 - RAC - Translated from the Pascal.                     *
*******************************************************************************/

static int com_carrier(void) {
        return com_installed && (inportb(uart_msr) & RLSD);
        }                                                   /* End com_carrier() */

/*******************************************************************************
*                           com_interrupt_driver()                              *
*******************************************************************************
* DESCRIPTION:        Handles communications interrupts.  The UART will interrupt *
*               whenever a character has been received or when it is ready      *
*               to transmit another character.  This routine responds by        *
*               sticking received characters into the receive queue and         *
*               yanking characters to be transmitted from the transmit queue    *
*                                                                               *
* REVISIOSN:  18 OCT 89 - RAC - Translated from the Pascal.                     *
*******************************************************************************/

static void interrupt com_interrupt_driver() {

        char    iir;                            /* Local copy if IIR */
        char    c;                              /* Local character variable */

/*  While bit 0 of the IIR is 0, there remains an interrupt to process  */

        while (!((iir = inportb(uart_iir)) & 1)) {      /* While there is an int ... */
        switch (iir) {                                  /* Branch on interrupt type */

                case 0:                                 /* Modem status interrupt */
                inportb(uart_msr);                      /* Just clear the interrupt */
                break;

                case 2:                                 /* Transmit register empty */

/*******************************************************************************
*  NOTE:  The test of the line status register is to see if the transmit    *
```

```
*       holding register is truly empty.  Some UARTS seem to cause      *
*       transmit interrupts when the holding register isn't empty,      *
*       causing transmitted characters to be lost.                      *
*************************************************************************/

                if (tx_chars <= 0)          /* If tx buffer empty, turn */
                        outportb(uart_ier,                  /*  off transmit interrupts */
                                inportb(uart_ier) & ~2);
                else {                                      /* Tx buffer not empty */
                        if (inportb(uart_lsr) & TXR) {
                        outportb(uart_data, tx_queue[tx_out++]);
                        if (tx_out == TX_QUEUE_SIZE)
                                tx_out = 0;
                        tx_chars--;
                        }
                        }                                           /* End 'tx buffer not empty */
                break;

                case 4:                                     /* Received data interrupt */
                c = inportb(uart_data);                     /* Grab received character */
                if (rx_chars < RX_QUEUE_SIZE) { /* If queue not full, save */
                        rx_queue[rx_in++] = c;   /*  the new character */
                        if (rx_in == RX_QUEUE_SIZE) /* Wrap index if needed */
                        rx_in = 0;
                        rx_chars++;                         /* Count the new character */
                        }                                   /* End queue not full */
                break;

                case 6:                                     /* Line status interrupt */
                inportb(uart_lsr);          /* Just clear the interrupt */
                break;

                }                                                   /* End switch */
        }                                                   /* End 'is an interrupt' */
        outportb(0x20, 0x20);                               /* Send EOI to 8259 */
        }                                                           /* End com_interrupt_driver() */

static void com_rx_wait()
 {
 time_out = TIME_OUT;
 while(com_rx_empty() && time_out--)
//          delay(2);
   delay(1);
 if(time_out < 1)
                puts("timed_out");
 }

/*
*************************************************************************
*
// Gordon Couger stuff here
*
*************************************************************************/
void error(char *a, char *b)
```

```
{
printf("error %s %s\   press any key /n",a,b);
getch();
}

static void strip_cr_linefeed(char *s)
        {
        while(*s)
         {
         if(*s == 10 || *s == 13)
                *s = 0;
         s++;
         }
        }

int send_command(char *command)
        {

        strip_cr_linefeed(command);

        com_tx_string(command);
//      delay(2);
        delay(1);
        com_tx(13);
//      delay(2);
        delay(1);
        return 0;
        }

int init_port(int port, int speed,int parity, int stop)
        {
         int value =0;

         if (0 !=(value = com_install(port)))
                 {
                 printf("com_install() error: %d\n", value);
                 error("comminstall","");
                 }
        com_raise_dtr();
        com_set_speed(speed);
        com_set_parity(parity, stop);
        atexit(com_deinstall);
        return value;
        }

void wait_for_response(char *b)
        {
        com_rx_wait();
//      while(com_rx_empty())
//        DONOTHING;
        while(0!=(*b = com_rx()))
         {
                 b++;
//               delay(2);
```

```
                delay(1);
                }
//          delay(2);
            delay(1);
            }
```

APPENDIX E

SUPERVISOR MODULE'S SOFTWARE (SUP4.C)

```
/*****************************************************************
*
*       Supervisor Module's Software
*       Created by: Enrique M. Acuña, Dec. 1995
*       File name: SUP4.C
*
*****************************************************************/

#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#include <dos.h>
#include "ibmcom3.c"     /* routines for low-level serial communications on the IBM PC */

#define mini(a,b,c)   min(min(a,b),c)

/* assignment of the numbers for the messages transmitted by the Supervisor */

#define SendToNav              1       /* command with desired veloc. and steering angle */
#define Req_Nav_RS_C_td        2       /* request of current speed, compass
                                          and rate of rotation */
#define Req_Nav_XYpos          3       /* request of current X and Y position */
#define Req_Vis_Son1to4        4       /* request of readings from sonars 1 to 4 */
#define Req_Vis_Son56          5       /* request of readings from sonars 5 and 6 */

/* PF numbers for the messages defined above */

#define RS_C_td_PF      0xED /* 237 */
#define XY_pos_PF       0xEC /* 236 */
#define Son1to4_PF      0xEB /* 235 */
#define Son56_PF        0xE9 /* 233 */

#define H               18.88   /* length from rear axis of the wheelchair to the front one */
#define R               8.06    /* radius of rear wheels of the wheelchair */

extern int time_out;            /* variable defined in IBMCOM3.C to set a time to account for a time
                                   out*/
int keypress;                   /* used for the keyboard interrupt */
int escape;                     /* used for the keyboard interrupt */

unsigned char bufTX[24]; /* buffer to hold the message to be transmitted */
unsigned char bufRX[24]; /* buffer to hold the message to be received */
unsigned char buf[24];   /* buffer to hold the data obtained after the joining process of ASCII
                            characters received from the SCS */
unsigned int temp;       /* temporal variable */
char i,j;                /* counters */
int kk;                  /* counter */

struct  time t;     /* to keep count of the time */

char n,l,r;         /* single ASCII character, left and right nibble that form the single ASCII
                       character */
```

```
int RoadSpeed=32767;     /* Desired velocity to be sent from supervisor to propulsion */
int SteeringAngle=32767; /* Desired steering angle to be sent from supervisor to propulsion */
int Range[7];            /* Six sonars' readings from vision */
int choice;              /* action or message (number) to send */
int OFF=1;               /* status of the emergency stop */

float ARoadSpeed,Compass,Comp_dot;     /* current speed, heading angle and rate of rotation */
float Xposition,Yposition;             /* current XY position */
double ASteeringAngle;                 /* current steering angle */

float clo(float, float, float);/* membership function for linguistic value Close */
float memfun(float, float, float);     /* membership function for linguistic value triangular shape */
float fa(float, float, float);         /* membership function for linguistic value Far */

void keypress_handler(void);           /* function to handle keyboard interrupts */
void CAN_interface(void);              /* function to implement serial communications */
void ascii(unsigned char);             /* see description below */
char i_ascii(unsigned char, unsigned char);  /* see description below */

void main(void)
{
#define SPEED 9600

int p,q,r;          /* counters */
float OldTime,CurTime,SamTime; /* to keep the time */

int nx1,nx2,nx3;                       /* number of linguistic values for each fuzzy input */
float mux1[3],mux2[3],mux3[3];         /* fuzzified input value */
float des_dist1,des_dist2,des_dist3; /* desired distances to be kept from sonars */
float son1,son2,son3;                  /* Range[] - distX */
float s1max,s2max,s3max;               /* input fuzzy sets maximum values */
float speedmax,stanglemax;             /* output fuzzy sets maximum values */

/* definition of fuzzy numbers for each linguistic value for input fuzzy sets */
float s1c,s1dp,s1f,s2vc,s2c,s2f,s3vc,s3c,s3f;
/* definition of half supports for input fuzzy sets */
float s1hsc,s1hsdp,s1hsf,s2hsvc,s2hsc,s2hsf,s3hsvc,s3hsc,s3hsf;

/* definition of fuzzy numbers for each linguistic value for output fuzzy sets */
float speedrm,speedrs,speedz,speedfs,speedfm;     /* for desired speed */
float stanr,stanst,stanl;                         /* for desired steering angle */
/* definition of half supports for output fuzzy sets */
float sphsm,sphss,sphsz;
float stahsr,stahsst,stahsl;

/* defintion of rule base's matrixes */
float rsp[3][3][3],rsps[3][3][3],rst[3][3][3],rsts[3][3][3];

/* firing strength */
float fi[3][3][3];

float sumNsp,sumDsp,sumNan,sumDan; /*temporal variables used in defuzzification */

init_port(2, SPEED,COM_NONE,1);
```

```
clrscr();

bufTX[0]=0x09;  /* 0x09 (buf[0]) starts mess to TX */

gettime(&t);
OldTime=t.ti_hour*3600+t.ti_min*60+t.ti_sec+t.ti_hund/100.0;

nx1=nx2=nx3=3;

des_dist1=90;
des_dist2=160;
des_dist3=160;

/* max. values */

s1max=40.0;
s2max=90.0;
s3max=90.0;
speedmax=30;
stanglemax=45;

/* input fuzzy sets and half of their support */

s1c=-s1max;
s1dp=0;
s1f=s1max;

s1hsc=s1max;
s1hsdp=s1max/4;
s1hsf=s1max;

s2vc=-s2max;
s2c=0;
s2f=s2max;

s2hsvc=s2max;
s2hsc=s2max/2;
s2hsf=s2max;

s3vc=-s3max;
s3c=0;
s3f=s3max;

s3hsvc=s3max;
s3hsc=s3max/2;
s3hsf=s3max;

/* output fuzzy sets and half of their support */

speedrm=-speedmax;
speedrs=-speedmax/2;
speedz=0;
speedfs=speedmax/2;
speedfm=speedmax;
```

```
sphsm=3*speedmax/4;
sphss=speedmax/2;
sphsz=speedmax/4;

stanr=-stanglemax;
stanst=0;
stanl=stanglemax;

stahsr=stanglemax;
stahsst=stanglemax/2;
stahsl=stanglemax;

/* rule base */
/* speed fuzzy numbers */
rsp[0][0][0]=speedrs; rsp[0][0][1]=speedrs; rsp[0][0][2]=speedrs;
rsp[0][1][0]=speedrs; rsp[0][1][1]=speedrs; rsp[0][1][2]=speedrs;
rsp[0][2][0]=speedrs; rsp[0][2][1]=speedrs; rsp[0][2][2]=speedrs;

rsp[1][0][0]=speedz; rsp[1][0][1]=speedz; rsp[1][0][2]=speedz;
rsp[1][1][0]=speedz; rsp[1][1][1]=speedz; rsp[1][1][2]=speedz;
rsp[1][2][0]=speedz; rsp[1][2][1]=speedz; rsp[1][2][2]=speedz;

rsp[2][0][0]=speedz; rsp[2][0][1]=speedz; rsp[2][0][2]=speedfs;
rsp[2][1][0]=speedz; rsp[2][1][1]=speedfs; rsp[2][1][2]=speedfs;
rsp[2][2][0]=speedfs; rsp[2][2][1]=speedfs; rsp[2][2][2]=speedfs;

/* speed fuzzy sets half supports */
rsps[0][0][0]=sphss; rsps[0][0][1]=sphss; rsps[0][0][2]=sphss;
rsps[0][1][0]=sphss; rsps[0][1][1]=sphss; rsps[0][1][2]=sphss;
rsps[0][2][0]=sphss; rsps[0][2][1]=sphss; rsps[0][2][2]=sphss;

rsps[1][0][0]=sphsz; rsps[1][0][1]=sphsz; rsps[1][0][2]=sphsz;
rsps[1][1][0]=sphsz; rsps[1][1][1]=sphsz; rsps[1][1][2]=sphsz;
rsps[1][2][0]=sphsz; rsps[1][2][1]=sphsz; rsps[1][2][2]=sphsz;

rsps[2][0][0]=sphsz; rsps[2][0][1]=sphsz; rsps[2][0][2]=sphss;
rsps[2][1][0]=sphsz; rsps[2][1][1]=sphss; rsps[2][1][2]=sphss;
rsps[2][2][0]=sphss; rsps[2][2][1]=sphss; rsps[2][2][2]=sphss;

/* steering angle fuzzy numbers */
rst[0][0][0]=stanst; rst[0][0][1]=stanst; rst[0][0][2]=stanl;
rst[0][1][0]=stanst; rst[0][1][1]=stanst; rst[0][1][2]=stanl;
rst[0][2][0]=stanr; rst[0][2][1]=stanr; rst[0][2][2]=stanst;

rst[1][0][0]=stanst; rst[1][0][1]=stanr; rst[1][0][2]=stanr;
rst[1][1][0]=stanl; rst[1][1][1]=stanst; rst[1][1][2]=stanst;
rst[1][2][0]=stanl; rst[1][2][1]=stanst; rst[1][2][2]=stanst;

rst[2][0][0]=stanst; rst[2][0][1]=stanr; rst[2][0][2]=stanr;
rst[2][1][0]=stanl; rst[2][1][1]=stanst; rst[2][1][2]=stanr;
rst[2][2][0]=stanl; rst[2][2][1]=stanl; rst[2][2][2]=stanst;

/* steering angle fuzzy sets half supports */
```

```
rsts[0][0][0]=stahsst; rsts[0][0][1]=stahsst; rsts[0][0][2]=stahsl;
rsts[0][1][0]=stahsst; rsts[0][1][1]=stahsst; rsts[0][1][2]=stahsl;
rsts[0][2][0]=stahsr; rsts[0][2][1]=stahsr; rsts[0][2][2]=stahsst;

rsts[1][0][0]=stahsst; rsts[1][0][1]=stahsr; rsts[1][0][2]=stahsr;
rsts[1][1][0]=stahsl; rsts[1][1][1]=stahsst; rsts[1][1][2]=stahsst;
rsts[1][2][0]=stahsl; rsts[1][2][1]=stahsst; rsts[1][2][2]=stahsst;

rsts[2][0][0]=stahsst; rsts[2][0][1]=stahsr; rsts[2][0][2]=stahsr;
rsts[2][1][0]=stahsl; rsts[2][1][1]=stahsst; rsts[2][1][2]=stahsr;
rsts[2][2][0]=stahsl; rsts[2][2][1]=stahsl; rsts[2][2][2]=stahsst;

choice = SendToNav;
CAN_interface();
delay(125);

escape=0;
do
{
        if (escape) {
                        RoadSpeed=32767;
                        SteeringAngle=32767;
                        OFF=0;
                        choice = SendToNav;
                        CAN_interface();
                        exit(1);
                }
        choice = Req_Vis_Son1to4;
        CAN_interface();
//      delay(125);

        son1=Range[0]-des_dist1;
        son2=Range[1]-des_dist2;
        son3=Range[2]-des_dist3;

        /* fuzzification */
        mux1[0]=clo(s1hsc,s1c,son1);
        mux1[1]=memfun(s1hsdp,s1dp,son1);
        mux1[2]=fa(s1hsf,s1f,son1);

        mux2[0]=clo(s2hsvc,s2vc,son2);
        mux2[1]=memfun(s2hsc,s2c,son2);
        mux2[2]=fa(s2hsf,s2f,son2);

        mux3[0]=clo(s3hsvc,s3vc,son3);
        mux3[1]=memfun(s3hsc,s3c,son3);
        mux3[2]=fa(s3hsf,s3f,son3);

        /* Process Logic */
        for (p=0;p<nx1;p++)
                for (q=0;q<nx2;q++)
                        for (r=0;r<nx3;r++)
                                fi[p][q][r]=mini(mux1[p],mux2[q],mux3[r]);
```

```
        /* defzzification */
        sumNsp=sumDsp=sumNan=sumDan=0;
        for (p=0;p<nx1;p++)
                for (q=0;q<nx2;q++)
                        for (r=0;r<nx3;r++)
                        {
                                sumNsp = sumNsp + rsp[p][q][r]*fi[p][q][r]/rsps[p][q][r];
                                sumDsp = sumDsp + fi[p][q][r]/rsps[p][q][r];
                                sumNan = sumNan + rst[p][q][r]*fi[p][q][r]/rsts[p][q][r];
                                sumDan = sumDan + fi[p][q][r]/rsts[p][q][r];
                        }

        RoadSpeed = (int)(sumNsp/sumDsp);

        if (RoadSpeed > speedmax)
                RoadSpeed = speedmax;
        if (RoadSpeed < (-speedmax))
                RoadSpeed = (-speedmax);

        RoadSpeed += 32767;

        SteeringAngle = (int)(sumNan/sumDan);

        if (SteeringAngle > stanglemax)
                SteeringAngle = stanglemax;
        if (SteeringAngle < (-stanglemax))
                SteeringAngle = (-stanglemax);

        SteeringAngle += 32767;

        choice = SendToNav;
        CAN_interface();
//      delay(125);



         keypress_handler();
         gettime(&t);
         CurTime=t.ti_hour*3600+t.ti_min*60+t.ti_sec+t.ti_hund/100.0;
         SamTime=CurTime-OldTime;
         OldTime=CurTime;
        gotoxy(1,5);
        printf("SamTime:  %4.2f  son1: %d des_vel: %d "
                        ,SamTime,Range[0],RoadSpeed-32767);
} while(!(choice==6));
         clrscr();
}

/* Parses a character (hexadecimal number) into two nibbles of 4 bits,
        left and right nibbles, and produces the corresponding ascii
        characters for each one of them ('0' to '9' and 'A' to 'F') */

void ascii(n)
unsigned char n;
```

```
{
l=((n>>4) & 0x0F) + 0x30;
r=(n & 0x0F) + 0x30;
if (l>0x39)
        l+=0x07;
if (r>0x39)
        r+=0x07;
}

/* takes two ascii characters ('0' to '9' and 'A' to 'F') and
        joins them into one single character (hexadecimal number
        from 0x00 to 0xFF) */

char i_ascii(ln,rn)
unsigned char ln,rn;
{
if (ln>0x39)
        ln-=7;
if (rn>0x39)
        rn-=7;
n=((ln-0x30) << 4) + (rn-0x30);
return n;
}

/* Manager of serial communications for the SCS and gateway to conform to the CAN */

void CAN_interface(void)
{
        CLEAR_BUFFERS;
         switch(choice) {
                case 1:
                        bufTX[1]=0x01;  /* indicates message #1 */
                        ascii(RoadSpeed>>8);
                        bufTX[2]=l;
                        bufTX[3]=r;
                        ascii(RoadSpeed);
                        bufTX[4]=l;
                        bufTX[5]=r;
                        ascii(SteeringAngle>>8);
                        bufTX[6]=l;
                        bufTX[7]=r;
                        ascii(SteeringAngle);
                        bufTX[8]=l;
                        bufTX[9]=r;
                        ascii(OFF>>8);
                        bufTX[10]=l;
                        bufTX[11]=r;
                        ascii(OFF);
                        bufTX[12]=l;
                        bufTX[13]=r;
                        send_command(bufTX);
                        wait_for_response(bufRX);

                        if (time_out < 1)
```

```
                    break;

                buf[0]=i_ascii(bufRX[2],bufRX[3]);
                buf[1]=i_ascii(bufRX[4],bufRX[5]);
                buf[2]=i_ascii(bufRX[6],bufRX[7]);
                buf[3]=i_ascii(bufRX[8],bufRX[9]);
                buf[4]=i_ascii(bufRX[10],bufRX[11]);
                buf[5]=i_ascii(bufRX[12],bufRX[13]);
                buf[6]=i_ascii(bufRX[14],bufRX[15]);
                buf[7]=i_ascii(bufRX[16],bufRX[17]);
                if (bufRX[1]==0x01) {  /* checks if correct mess is received */
                        gotoxy(1,16);
                        printf("Acknowledged: %x %x %x %x %x %x %x
                        %x",buf[0],buf[1],buf[2],buf[3],buf[4],buf[5],buf[6],buf[7]);
                        break;
                        }
                else {
                        gotoxy(1,16);
                        printf("Not_Acknowledged");
                        break;
                        }
        case 2:
                bufTX[1]=0x02;  /* indicates message #2 */
                bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
                bufTX[3]=0x30;
                bufTX[4]=0x45;  /* bufTX[4] and [5] send RS_C_td_PF (0xED) */
                bufTX[5]=0x44;  /* in ascii */
                bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
                bufTX[7]=0x30;
                gotoxy(1,18);
                printf("RS_C_td requested   ");
                send_command(bufTX);
                wait_for_response(bufRX);

                if (time_out < 1)
                        break;

                buf[0]=i_ascii(bufRX[2],bufRX[3]);
                buf[1]=i_ascii(bufRX[4],bufRX[5]);
                buf[2]=i_ascii(bufRX[6],bufRX[7]);
                buf[3]=i_ascii(bufRX[8],bufRX[9]);
                buf[4]=i_ascii(bufRX[10],bufRX[11]);
                buf[5]=i_ascii(bufRX[12],bufRX[13]);
                if (bufRX[1]==0x02) {
                        temp=(buf[0]<<8) + buf[1];
                        ARoadSpeed=(temp-32767.0)/10;

/*                      printf("buf[0:1] %x %x buf[2,3] %x
                         %x",buf[0],buf[1],buf[2],buf[3]);
                        printf("buf[4,5] %x %x ",buf[4],buf[5]);
*/                      temp=(buf[2]<<8) + buf[3];
                        Compass=temp-32767.0;
                        temp=(buf[4]<<8) + buf[5];
                        Comp_dot=(temp-32767.0)/10;
```

```
//                      ASteeringAngle=atan(Comp_dot*H/(ARoadSpeed*R));
                        ARoadSpeed*=0.0215;
                        gotoxy(1,19);
                        printf("Road Speed: %5.2f",ARoadSpeed);
/*                      gotoxy(1,20);
                        printf("Compass:   %5.2f",Compass);
                        gotoxy(1,21);
                        printf("AS.Angle:  %lf ",Comp_dot);
*/                      break;
                        }
                else {
                        gotoxy(1,19);
                        printf("No_expected_response");
                        break;
                        }
        case 3:
                bufTX[1]=0x03;  /* indicates message #3 */
                bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
                bufTX[3]=0x30;
                bufTX[4]=0x45;  /* bufTX[4] and [5] send XY_pos_PF (0xEC) */
                bufTX[5]=0x43;  /* in ascii */
                bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
                bufTX[7]=0x30;
                gotoxy(1,23);
                printf("XY_pos requested  ");
                send_command(bufTX);
                wait_for_response(bufRX);

                if (time_out < 1)
                        break;

                buf[0]=i_ascii(bufRX[2],bufRX[3]);
                buf[1]=i_ascii(bufRX[4],bufRX[5]);
                buf[2]=i_ascii(bufRX[6],bufRX[7]);
                buf[3]=i_ascii(bufRX[8],bufRX[9]);
                if (bufRX[1]==0x03) {
                        Xposition=((buf[0]<<8) + buf[1])-32767;
                        Yposition=((buf[2]<<8) + buf[3])-32767;
                        gotoxy(1,24);
                        printf("X-Y position %5.2f %5.2f ",Xposition,Yposition);
                        break;
                        }
                else {
                        gotoxy(1,24);
                        printf("No_expected_response");
                        break;
                        }
        case 4:
                bufTX[1]=0x04;  /* indicates message #4 */
                bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
                bufTX[3]=0x30;
                bufTX[4]=0x45;  /* bufTX[4] and [5] send Son1to4_PF (0xEB) */
                bufTX[5]=0x42;  /* in ascii */
                bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
```

```
        bufTX[7]=0x30;
        gotoxy(35,13);
        printf("Son1 to 4 requested   ");
        send_command(bufTX);
        wait_for_response(bufRX);

        if (time_out < 1)
                break;

        buf[0]=i_ascii(bufRX[2],bufRX[3]);
        buf[1]=i_ascii(bufRX[4],bufRX[5]);
        buf[2]=i_ascii(bufRX[6],bufRX[7]);
        buf[3]=i_ascii(bufRX[8],bufRX[9]);
        buf[4]=i_ascii(bufRX[10],bufRX[11]);
        buf[5]=i_ascii(bufRX[12],bufRX[13]);
        buf[6]=i_ascii(bufRX[14],bufRX[15]);
        buf[7]=i_ascii(bufRX[16],bufRX[17]);
        if (bufRX[1]==0x04) {
                Range[0]=(buf[0]<<8) + buf[1];
                Range[1]=(buf[2]<<8) + buf[3];
                Range[2]=(buf[4]<<8) + buf[5];
                Range[3]=(buf[6]<<8) + buf[7];
                gotoxy(35,14);
                printf("Sonar 1-2: %d  %d  ",Range[0],Range[1]);
                gotoxy(35,15);
                printf("Sonar 3-4: %d  %d  ",Range[2],Range[3]);
                break;
                }
        else {
                gotoxy(35,14);
                printf("No_expected_response");
                break;
                }
case 5:
        bufTX[1]=0x05;  /* indicates message #5 */
        bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
        bufTX[3]=0x30;
        bufTX[4]=0x45;  /* bufTX[4] and [5] send Son56_PF (0xE9) */
        bufTX[5]=0x39;  /* in ascii */
        bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
        bufTX[7]=0x30;
        gotoxy(35,17);
        printf("Sonar 5-6 requested   ");
        send_command(bufTX);
        wait_for_response(bufRX);

        if (time_out < 1)
                break;

        buf[0]=i_ascii(bufRX[2],bufRX[3]);
        buf[1]=i_ascii(bufRX[4],bufRX[5]);
        buf[2]=i_ascii(bufRX[6],bufRX[7]);
        buf[3]=i_ascii(bufRX[8],bufRX[9]);
        if (bufRX[1]==0x05) {
```

```
                                Range[4]=(buf[0]<<8) + buf[1];
                                Range[5]=(buf[2]<<8) + buf[3];
                                gotoxy(35,18);
                                printf("Sonar 5-6: %d  %d  ",Range[4],Range[5]);
                                break;
                                }
                        else {
                                gotoxy(35,18);
                                printf("No_expected_response");
                                break;
                                }
                case 6:
                        exit(1);
                        break;
                default:
                        gotoxy(1,19);
                        printf("Error");
                        break;
                }
}

/*************************************************************************
 *                                      *
 *       KEYPRESS MAIN HANDLER                                           *
 *                                      *
 *************************************************************************/
void keypress_handler(void)
{
        static cc;

        if(kbhit())
        {
                cc = getch();  /* escape */
                switch (cc) {
                case 27:
                        escape=1;
                        break;
/*              case 'c':
                        action1();
                        break;
*/              default:
                        printf("error");
                        break;
                }
        }
}

float clo(float halsup,float x0, float x)
{
if (x<=x0) return 1.0;
if (x>=(x0+halsup)) return 0.0;
return (1.0-(x-x0)/halsup);
}
```

```
float memfun(float halsup,float x0, float x)
{
if (x<=(x0-halsup)) return 0.0;
if (x>=(x0+halsup)) return 0.0;
return (1.0-abs(x-x0)/halsup);
}

float fa(float halsup,float x0, float x)
{
if (x>=x0) return 1.0;
if (x<=(x0-halsup)) return 0.0;
return (1.0-(x0-x)/halsup);
}
```

APPENDIX F

SUPERVISOR MODULE'S SOFTWARE (ZZ_SUP4.PRJ)

COMMENTS TO THIS APPENDIX

This appendix only includes the most important files that can be used in the future work of implementing the Supervisor based on the work of Andujar and Shanley, the rest of the files are stored in disk.

What is most important when implementing such Supervisor is to make sure of including the corrections in the values of X and Y position and heading angle (to avoid the problem of the lack of floating point numbers); in other words,

if x, y and Θ are the values coming from the gateway (indirectly from the propulsion module) and X, Y and θ are supposed to be the corresponding values in the Spervisor then the following equations must be implemented:

$$X = X + x*\cos(\Theta + \theta)$$

$$Y = Y + y*\cos(\Theta + \theta)$$

$$\theta = \theta + \Theta$$

and in the propulsion module:

```
x = offset;
y = offset,
Θ = offset;
```

```
/*************************************************************************
*
*       This file pretends to show the sequence of Shanley's code by listing the main
*       functions in each stage
*       prepared by Enrique M. Acuña as a guide to organize the original code
*       file name: ZZ_DOC1.CPP
*
*************************************************************************/

/********** Installs Communication Servers for Each Module. **********/
ZZ_CAN_InstallServer(SUPERVISOR,ZZ_SuperServer);
                ZZ_SuperServer

ZZ_CAN_InstallServer(NAVIGATION,ZZ_NavServer);
                ZZ_NavServer

ZZ_CAN_InstallServer(VISION,ZZ_VisionServer);
                ZZ_VisionServer

/*********       Initializes Each Module          **********/
ZZ_InitSuper()
                * use graphics
                * PASS MEMORY ADDRESSES OF VARIABLES USED BY SUPERVISOR SUB-
MODULES
        ZZ_SupertoMap;
         *      GET MAIN SUPERVISOR PRIVATE VARIABLE ADDRESSES USED BY  *
         *      MAPPING FUNCTIONS AND ASSIGN THEM TO LOCAL PRIVATE      *
         *      VARIABLES.                                              *
         *                                      *
        ZZ_SupertoConsl;
         *      OBTAINS MEMORY ADDRESSES FOR WAYPOINT LOCATION VARIABLES *
        ZZ_InitGraph();
        ZZ_InitMap();
         *      INITIALIZE MAP TO ALL OCCUPIED SPACE.           *
         * ------------------------------------------------------*
         *      NOTE: ZZ_DrawRoom is used only for simulation purposes. *
         *          ZZ_DrawCar is not necessary, it is used to locate  *
         *          the robot on the screen.                *


ZZ_InitVision();
***       THE FOLLOWING STATEMENTS IS FOR SONAR SIMULATION
***     PURPOSES ONLY. WHEN FINALLY IMPLEMENTED THE FOLLOWING
***       STATEMENT SHOULD BE ERASED !

ZZ_InitNav();
        ZZ_Cart2Polar

ZZ_DrawCursor();

while(1) {

        /*********      Propulsion Loop
         *********      ---------------------------------------
```

```
        *********       For every Supervisor Loop, the Propulsion
        *********       Module Loops Four Times with 0.05 sample period
        *********       for the actuator controls.
        *********/
ZZ_NavLoop();
        ZZ_UpdateSensorMeas();
        *       UPDATES ALL ACTUATORS, POSITION AND BEARING              *
        *       INFORMATION                                              *
                *
        ****** 'ZZ_CarSim' is for Simulation Purposes Only.
        ****** Function used TO OBTAIN SENSOR INFO Goes Here
                ZZ_CarSim();
                *****   Following instructions are to be replaced with data acquisition*
                *****   MotorTorque is the input to the DC-Motor
                *****   Input to Steering is to be determined based on Steering Control
                *****   Mechanism. Simulation is just assuming instantaneous control.
                ZZ_CheckCrash(void); /* uses graphics */
        ZZ_UpdateActuators();
        *       ALL THE CONTROLS FOR THE ACTUATORS GO HERE               *
        *****   Following instructions ... as for ZZ_CarSim

ZZ_VisionLoop();
        *       SONAR SENSOR MEASUREMENT GOES HERE                       *
        ZZ_UpdateSensorMeas();
        *       Following function to be replaced with data acquisition
        ZZ_SonarSim();   /* uses graphics */

ZZ_SuperLoop();
        ZZ_DrawCar();
        ZZ_DrawCursor();
        ZZ_GetSensorData();
        if(--FLAG<0)
        {
                ZZ_UpdateMap();
                FLAG = start;
        }
        ZZ_DrawCursor();
        ZZ_DrawCar();
        ZZ_ReachTarget();
        ZZ_ReachWayPoint();
        ZZ_CollisionAvoidance();
        ZZ_SendToNav();

keypress_handler();


THE END
}
```

```
/*****************************************************************
*
*       This file pretends clarify the sequence of events that are managed by the
*       the Supervisor Module in Shanley's code
*       prepared by Enrique M. Acuña as a guide to understand and decide what
*       material can be reused for implementation in the real AV
*       file name: ZZ_DOC2.CPP
*
*****************************************************************/


ZZ_SuperLoop() {

        ZZ_DrawCar();

        ZZ_DrawCursor();

        ZZ_GetSensorData();     /* in file: ZZ_SUPR2.C */
                ZZ_CanRequest(Navigation)
                ZZ_CanRequest(Vision)
                Real2Screen()
                Polar2Cart()
                Polar2Cart()

        if(--FLAG<0)
        {
                ZZ_UpdateMap();       /* in file: ZZ_MAP.C */
                        Screen2Real()
                        xy_to_bit()
                        clear_row()

                FLAG = start;
        }
        ZZ_DrawCursor();

        ZZ_DrawCar();

*       ZZ_ReachTarget();       /* study this function in file ZZ_SUPR2.C */
                isempty()   /* returns a 1 if queue is empty, otherwise returns 0*/
                        dequeue(q,x)
                        *       RETURNS THE FIRST ELEMENT IN QUEUE IN *x AND ERASES     *
                        *       IT FROM THE QUEUE.                                      *
                        Screen2Real()
                gotoxy()            /*graphics*/
                printf()
                Real2Screen()
                if(!ZZ_Uncovered(TX,TY,4))  /* returns 0 or 1; file: ZZ_MAP.C */
                        getbit(x,y) /* return the value of the bit x,y; either '0' or '1'*/
                {
                        if(METHOD == FUZZY)
                                METHOD = PLANNER;
                        else
                                METHOD = FUZZY;
                ..... }
```

```
        resetqueue()        /* empties queue */

*       Planner()           /* study in file: ZZ_PLANR.C */
                * Set Target as final destination
                * Set current position as initial point

                enqueue()  /* addes *x to the end of the queue */
                        gotoxy()
                        farmalloc()
                        farfree()

                while(1)
                        BASIC PLANNER
                        lineofsight()
                *       ENQUEUS *X LOCATION IF LINE OF SIGHT IS POSSIBLE *
                        * BETWEEN LAST WAYPOINT IN QUEUE AND *X
*
                                getbit()
                        * GO AROUND OBSTACLE COUNTERCLOCKWISE  AND
                        * CLOCKWISE UNTIL REACHING LEAVE POINT.
                        ZZ_FollowWallClockwise(&Cw);
                                getbit()
                        ZZ_FollowWallCounterClockwise(&Ccw);
                                getbit()

                        enqueue()
                                endtotarget()

                        setcolor()
                        qplot()    /* in file: ZZ_QUEUE.C uses graphics */
                        optimize()  /* in file: ZZ_QUEUE.C

                *       OPTIMIZES GENERATED PATH BY ELIMINATING UNNECESARY
                                *       WAYPOINTS.  */                          *

                        dequeue()
                        Screen2Real()

                Real2Screen()
                Uncovered()
                LimitAngle()  /* limits angle to -180 to 180 */
                Range()     /* RETURNS DISTANCE FROM ZERO COORDINATE.
                                                        in ZZ_MISC.c file */

        ZZ_ReachWayPoint();
                * CONTROLLER # 1
                * TRACK CURRENT WAYPOINT
                * FUZZY INFERENCE RULES

        ZZ_CollisionAvoidance();
                CONTROLLER # 2
                COLLISION AVOIDANCE AND
                GET OUT OF TIGHT SPOTS
                FUZZY INFERENCE RULES
```

```
            Mim()
            Max()
            LimitAngle()

        ZZ_SendToNav();
            Command()
}
```

```
/*****************************************************************
*
*       This file contains all the CAN related items of Shanley's code, this file's
*       information can be used to decide where and how to cut or disassemble the
*       original simulation code and take only what the Supervisor will need to be
*       to be implemented in the real AV
*       prepared by Enrique M. Acuña
*       file name: ZZ_DOC2.CPP
*
*****************************************************************/

#define TERMINALMAX 255
/*****************************************************************
 *                                   *
 *       INITIALIZING MEMORY SPACE FOR CAN TERMINALS          *
 *                                   *
 *****************************************************************/
static void (*ZZ_Terminal[TERMINALMAX])(byte SourceAddress,
                        int *DataContent,double *data,byte *datanum);
/*****************************************************************
 *                                                               *
 *       INSTALLS TERMINAL ON CAN
          *
 *                                                               *
 *****************************************************************/
int     ZZ_CAN_InstallServer(byte SourceAddress,
                void (*TempName)(byte SourceAddress,int *DataContent,
                                        double *data,byte *datanum))
{
        if (ZZ_Terminal[SourceAddress]==0L)
        {
                ZZ_Terminal[SourceAddress] = TempName;
                return(TERMINALINSTALLED);
        }
        return(OCCUPIED);
}
/*****************************************************************
 *                                                               *
 *       COMMUNICATION REQUEST THROUGH NETWORK                        *
 *                                                               *
 *****************************************************************/
void    ZZ_CAN_Request(byte Priority,byte SourceAddress,
                byte DestinationAddress,int *DataContent,
                                        double *data,byte *datanum)
{
        *DataContent = REQUEST;
        if (ZZ_Terminal[DestinationAddress])
        ZZ_Terminal[DestinationAddress](SourceAddress,
                                        DataContent,data,datanum);
}
/*****************************************************************
 *                                                               *
 *       SEND A REFERENCE COMMAND SIGNAL THROUGH CAN                  *
 *                                                               *
```

```
*************************************************************************/
void    ZZ_CAN_Command(byte Priority,byte SourceAddress,
            byte DestinationAddress,int *DataContent,
                                        double *data,byte *datanum)
{
        *DataContent = COMMAND;
        ZZ_Terminal[DestinationAddress](SourceAddress,
                                        DataContent,data,datanum);
}

/********* Installs Communication Servers for Each Module. **********/
        ZZ_CAN_InstallServer(SUPERVISOR,ZZ_SuperServer);
        ZZ_CAN_InstallServer(NAVIGATION,ZZ_NavServer);
        ZZ_CAN_InstallServer(VISION,ZZ_VisionServer);

/* ZZ_SUPR2.C */

void    ZZ_GetSensorData(void)
        int i=1;
        ZZ_CAN_Request(HIGHPRIORITY,SUPERVISOR,VISION,&DataContent,
                                                Range,&DataNum);
        ZZ_CAN_Request(HIGHPRIORITY,SUPERVISOR,NAVIGATION,&DataContent,
                                                Data,&DataNum);
        i=1;
        ARoadSpeed      = Data[i++];
        Compass         = Data[i++];
        Xposition       = Data[i++];
        Yposition       = Data[i++];
        ASteeringAngle  = Data[i++];
        ABrake          = Data[i++];

void    ZZ_SendToNav(void)
{
        int i=1;
        Data[i++] = RoadSpeed;
        Data[i++] = SteeringAngle;
        Data[i++] = OFF;
        DataNum = 2;
        ZZ_CAN_Command(HIGHPRIORITY,SUPERVISOR,NAVIGATION,&DataContent,
                                            Data,&DataNum);
}
/*************************************************************************
*                                                                        *
*       HANDLES ALL NETWORK REQUESTS FROM SUPERVISOR                     *
*       AND VISION MODULES.                                              *
*                                                                        *
*************************************************************************/
void ZZ_NavServer(byte SourceAddress,int *DataContent,
                                        double *Data, byte *DataNum)
{
        switch(*DataContent)
        {
        case REQUEST:
                *DataContent = RS_SA_X_Y_COMP_B;
```

```
                Data[1] = RoadSpeed2;
                Data[2] = Compass2;
                Data[3] = Xposition2;
                Data[4] = Yposition2;
                Data[5] = ASteeringAngle2;
                Data[6] = Brake;
                *DataNum = 6;
        break;
        case COMMAND:
                *DataContent = SUCCESS;
                CruiseSpeed = Data[1];
                DSteeringAngle = Data[2];
                Brake = (int)Data[3];
                *DataNum = 0;
        break;
        }
}
/*********************************************************************
 *   This Function handles communication requests from          *
 *                  other Modules through                       *
 *          Controller Area Network                             *
 *********************************************************************/
void ZZ_SuperServer(byte SourceAddress,int *DataContent,
                                        double *Data, byte *DataNum)
{
        *DataContent = NOTALLOWED;
}

/*********************************************************************
 *                                                                   *
 *        VISION COMMUNICATION NETWORK HANDLER                       *
 *                                                                   *
 *********************************************************************/
void ZZ_VisionServer(byte SourceAddress,int *DataContent,
                                        double *Data, byte *DataNum)
{
        int i;
        switch(*DataContent)
        {
        case REQUEST:
                *DataContent = SIXSENSORS_2CROSSED;
                for(i=0;i<NUM_SENSORS;i++)
                        Data[i] = Range2[i];
                *DataNum = NUM_SENSORS;
        break;

        case COMMAND:
                *DataContent = SUCCESS;
                EMERGENCY = Data[1];
                *DataNum = 0;
        break;
        }
}
```

```
/*****************************************************************
*
*       This file shows all the included files needed by the different files that form
*       Shanley's simulation software. With this file it is known the relationship
*       between files
*       prepared by Enrique M. Acuña
*       file name: ZZ_DOCIN.CPP
*
*****************************************************************/

 FILE            : ZZ_CAN.C

#include "ZZ_CAN.H"

 FILE            : ZZ_CONSL.C

#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#include <stdio.h>
#include <bios.h>                       /* Microsoft specific */

 FILE            : ZZ_FUZZY.C

#include"ZZ_fuzzy.h"
#include<alloc.h>

 FILE            : ZZ_GRAPH.C

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
#include"ZZ_CAN.H"
#include"ZZ_MISC.H"
#include"ZZ_SUPR2.H"
#include"ZZ_GRAPH.H"

 FILE            : ZZ_MAN.C

#include<graphics.h>
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<alloc.h>
#include"ZZ_CAN.H"
#include"ZZ_SUPR2.H"
#include"ZZ_VSION.H"
#include"ZZ_NAV.H"
#include"ZZ_OBJCT.H"
#include"ZZ_CONSL.H"

 FILE            : ZZ_MAP2.C
```

```
#include<graphics.h>
#include<time.h>
#include<stdlib.h>
#include<math.h>
#include <conio.h>
#include"ZZ_CAN.H"
#include"ZZ_SUPR2.H"
#include"ZZ_CARSP.H"
#include"ZZ_MAP.H"

 FILE            : ZZ_MISC.C

#include<math.h>

 FILE            : ZZ_NAV.C

#include<math.h>
#include<graphics.h>
#include "ZZ_CAN.H"
#include"ZZ_MISC.H"
#include"ZZ_GRAPH.H"
#include "ZZ_CARSP.H"
#include "ZZ_SUPR2.H"
#include "ZZ_NAV.H"

 FILE            : ZZ_OBJCT.C

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

 FILE: ZZ_PLANR.c
 ***************************/

#include<graphics.h>
#include<alloc.h>

#include"ZZ_QUEUE.H"

 FILE            : ZZ_QUEUE.C

#include<stdlib.h>
#include<alloc.h>
#include<conio.h>
#include<graphics.h>
#include<stdio.h>
#include<math.h>
#include "zz_queue.h"

 FILE            : ZZ_SUPR2.C

#include<alloc.h>
```

```
#include<math.h>
#include<stdlib.h>
#include<graphics.h>
#include<stdio.h>
#include"ZZ_CAN.H"
#include"ZZ_CARSP.H"
#include"ZZ_MISC.H"
#include"ZZ_MAP.H"
#include"ZZ_GRAPH.H"
#include"ZZ_SUPR2.H"
#include"ZZ_QUEUE.H"

 FILE              : ZZ_VSION.C

#include<alloc.h>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
#include"ZZ_GRAPH.H"
#include"ZZ_OBJCT.H"
#include"ZZ_CARSP.H"
#include"ZZ_CAN.H"
#include"ZZ_VSION.H"
```

```
/*****************************************************************
*
*       This file contains all header files of Shanley's code using Gaussian fuzzy sets
*       prepared by Enrique M. Acuña
*       file name: ZZ_DOC.H
*
*****************************************************************/
/*ZZ_CAN.h*/

#define OCCUPIED    252
#define TERMINALINSTALLED 251

/*                          PRIORITIES */
#define HIGHPRIORITY  6
#define MEDIUMPRIORITY 3
#define LOWPRIORITY    1

/***                         ADDRESSES ***/
#define VISION    3
#define NAVIGATION 2
#define SUPERVISOR 1


#define BROADCAST  4
#define NOTALLOWED 127
#define REQUEST    128
#define COMMAND    129
#define SUCCESS    130

#define RS_SA_X_Y_COMP_B 1
#define SIXSENSORS_2CROSSED 2


#define RS_XPOSITION 1.2
#define RS_YPOSITION 1.2
#define RS_XCOMPASS 1.2
#define RS_ROADSPEED 1.2
#define RS_STEERANGLE 1.2
#define RS_ROADRANGEC1 1.2
#define RS_ROADRANGEC2 1.2
#define ON 1
#define OFF 0

typedef unsigned short byte;

static int DataContent;

static double Data[20];

static     byte DataNum;


void       ZZ_CAN_Request(byte Priority,byte SourceAddress,
```

```
                byte DestinationAddress,int *DataContent,
                                        double *data,byte *datanum);

void    ZZ_CAN_Command(byte Priority,byte SourceAddress,
                byte DestinationAddress,int *DataContent,
                                        double *data,byte *datanum);

int     ZZ_CAN_InstallServer(byte SourceAddress,
                void (*TempName)(byte SourceAddress,int *DataContent,
                                        double *data,byte *datanum));

/*ZZ_CARSP.H */

static double


        /*********************************************************
         *
         *      Sensor angle relative to forward direction
         *      Positive angles correspond to clockwise direction
         *
         *********************************************************/
        NominalAngle[]=         {0,M_PI,M_PI_4*1.5,-M_PI_4*1.5,-1.*M_PI_2,1.*M_PI_2},


        /*******************************************************
         *
         *      Sensor position in meters relative to center
         *      of front axis
         *
         *******************************************************/
        Xr[]={0,0,-.2,.2,-.25,.25},
        Yr[]={0,.5,0,0,.45,.45},

        /*******************************************************
         *
         *      Car edges in meters relative to center
         *      of front axis
         *
         *******************************************************/
        CarEdgeX[]={-.2,-.2,.2,.2},
        CarEdgeY[]={.5,0,0,.5},

        AxleToAxleLength=.5,
        MotorInertia=.5,
        MotorViscosity=.05;

/* ZZ_CONSL.C */

void    keypress_handler(void);
extern void print_map_handler(void);
void    kbsig(void);
```

```
/*************************************************************************
*************************************************************************
                        FUZZY LIBRARY INCLUDE FILE

        Fuzzy Inference Library
        Copyright 1993
        Ricardo Andujar
*************************************************************************/

/*************************************************************************
********
********                MEMBERSHIP TYPE DEFINITIONS
********
*************************************************************************/
typedef struct
        {
                double  bottomleft,
                        topleft,
                        topright,
                        bottomright;
        } ZZ_TRAPEZOID;

typedef struct
        {
                int     discretenum;
                double  lowrange,
                        highrange,
                                *discrete;
        } ZZ_FUZZYOUTPUT;

/*************************************************************************
********
********    FUZZY LIBRARY FUNCTIONS FOR ALL TYPE OF INPUTS
********
*************************************************************************/
void    ZZ_InitFuzzyOutput(double lowrange,double highrange,
                        ZZ_FUZZYOUTPUT *,int discretenum);
void    ZZ_DelFuzzyOutput(ZZ_FUZZYOUTPUT *);
void    ZZ_AddMax(double min,ZZ_TRAPEZOID *,ZZ_FUZZYOUTPUT *);
double  ZZ_Defuzzify(ZZ_FUZZYOUTPUT *);
void    ZZ_ClearFuzzyOutput(ZZ_FUZZYOUTPUT *);

/*************************************************************************
********
********    FUZZY LIBRARY FUNCTIONS FOR SINGLETON INPUTS
********
*************************************************************************/
double  ZZ_Member(double,ZZ_TRAPEZOID *);
double  ZZ_Max(double,double);
double  ZZ_Min(double,double);
double  ZZ_Complement(double,ZZ_TRAPEZOID *);

/* ZZ_GRAPH.H */
```

```
void    ZZ_InitGraph(void);
void    ZZ_CloseGraph(void);
void    ZZ_DrawBlackBox(void);
void    ZZ_Polar2Cart(double ang,double r,double *x,double *y);
void    ZZ_Cart2Polar(double x,double y,double *ang,double *r);
void    ZZ_Real2Screen(double x,double y,int *xc, int *yc);
void    ZZ_Screen2Real(int xc,int yc,double *x, double *y);

/* ZZ_MAP.H */

void    ZZ_DrawRoom(void);
void    ZZ_DrawGrayArea(void);
void    ZZ_DrawBlackBox(void);
void    ZZ_DrawCar(void);
void    ZZ_UpdateMap(void);
int     ZZ_Uncovered(int TX, int TY,int res);
void    ZZ_InitMap(void);
void print_map_handler(void);

/* Structure defines a bit as an integer which
 * provides an x-y location in a multi-
 * dimensional field of bits
 */

typedef struct
        {
        int col_bit;
        int row_bit;
        }bit;


int getbit(int x,int y);
bit xy_to_bit(float, float);
void clear_row(bit, bit,int,int);

/* ZZ_MISC.H */

void    ZZ_Swap(double *one,double *two);
double  ZZ_Range(double x,double y);
double  ZZ_Atan2(double y, double x);
void    ZZ_LimitAngle(double *Angle);

/* ZZ_NAV.H */

void    ZZ_NavLoop(void);
void ZZ_NavServer(byte SourceAddress,int *DataContent,
                                        double *Data, byte *DataNum);

/* ZZ_OBJCT.H */

void    ZZ_CreateCircle(int x,int y);
void    ZZ_DestroyCircle(void);
void    ZZ_EraseCircle(void);
void    ZZ_DrawCircle(void);
```

```
/* ZZ_QUEUE */

typedef struct
        {
                int     x,
                        y;
        } ZZ_Point;



typedef ZZ_Point DATA;

struct  Linked_list
{
        DATA  d;
        struct  Linked_list     *next,*previous;
};


typedef  struct  Linked_list     ELEMENT;
typedef ELEMENT                          *LINK;

struct queue
{
        LINK   front,
               rear;
};

typedef struct queue       QUEUE;

/* ZZ_SUPR2.H */

int     qsearch(QUEUE *q, DATA *x);
int     isempty(QUEUE *q);
DATA    vfront(QUEUE *q);
void    dequeue(QUEUE *q, DATA *x);
void    enqueue(QUEUE *q, DATA *x);
void    qplot(QUEUE *q);
int     optimize(QUEUE *q);
int     sumqueue(QUEUE *q);
int     numqueue(QUEUE *q);
int     lineofsight(QUEUE *q, DATA *x, int *x3, int *y3, int *x4, int *y4);
int     endtotarget(QUEUE *q, DATA *x);
void    unqueue(QUEUE *q, DATA *x);
void    resetqueue(QUEUE *q);

#define SCREENCONVERT     40.0                  /* (pixels/meter) */
#define BOTTOMLIMIT 479
#define TOPLIMIT    25
#define RIGHTLIMIT  639
#define LEFTLIMIT   1
#define SMAXRANGE   400
```

```
#define  HALFSPREADANGLE 10.0/180.0*M_PI   /* 10 Degrees */
#define NUM_SENSORS 6
#define  NUM_EDGES    4


void      ZZ_InitSuper(void);
void      ZZ_EraseSuper(void);
void      ZZ_SuperLoop(void);
void      ZZ_SuperServer(byte SourceAddress,int *DataContent,
                                              double *Data, byte *DataNum);

/* ZZ_VISION */

#define ON 1
#define ROADRANGEC1 1.4
#define ROADRANGEC2 3.2
#define SPEED_SOUND 343.0   /* (meters/sec)  */
#define SCREENCONVERT 40.0   /* (pixel/meter) */
#define SMAXRANGE    400
#define  HALFSPREADANGLE  10.0/180.0*M_PI  /*10 Degrees*/
#define NUM_SENSORS 6



void ZZ_VisionServer(byte SourceAddress,int *DataContent,
                                              double *Data, byte *DataNum);

void      ZZ_VisionLoop(void);
```

```
/*************************************************************************
 FILE            : ZZ_MAN.C

 DESCRIPTION  : MAIN LOOP SIMULATION FOR AUTONOMOUS VEHICLE

                  This file contains main loop for simulation.
-------------------------------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE     : MARCH 22, 1993
*************************************************************************/
/*
*       Modified to test supervisor with the AV on the bench with the sonar information
*       simulated
*       Prepared by Enique M. Acuña, Oct. 1995
*/
/*************************************************************************
 INCLUDE FILES FOR MAN
 *************************************************************************/
//#include<graphics.h>
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<alloc.h>
#include"ZZ_CAN.H"
#include"ZZ_SUPR2.H"
#include"ZZ_VSION.H"
#include"ZZ_NAV.H"
#include"ZZ_OBJCT.H"
//#include"ZZ_CONSL.H"

extern int keypress,escape;
int super,navigation,vision,graph;

void    main(void)
{
        int i;


/********* Installs Communication Servers for Each Module. **********/
        ZZ_CAN_InstallServer(SUPERVISOR,ZZ_SuperServer);
        ZZ_CAN_InstallServer(VISION,ZZ_VisionServer);

/*********       Initializes Each Module         **********/
        ZZ_InitSuper();
        ZZ_InitVision();
        ZZ_InitNav();

/*************************************************************************
 *********       Main Program Loop
 ********* ---------------------------------------------------------------
 *********       When finally impleneted, each module will have it's own
 *********       separate loop on different processors.
```

```
**************************************************************************/
        while(1)
        {
        /*********     Propulsion Loop
         *********     ---------------
         *********     For every Supervisor Loop, the Propulsion
         *********     Module Loops Four Times with 0.05 sample period
         *********     for the actuator controls.
         *********/

        /********* Vision Loop              *********/
                        ZZ_VisionLoop();

        /********* Supervisor Loop          *********/
                        ZZ_SuperLoop();

        /*********   Handles Keyboard Presses  **********/
//              keypress_handler();
        }
}
```

```
/*************************************************************************
 FILE            : ZZ_SUPR2.C

 DESCRIPTION     : SUPERVISOR MODULE FOR AUTONOMOUS VEHICLE

                   This file contains code for the fuzzy controller,
                   and high level reasoning.
-------------------------------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE     : MARCH 22, 1993
*************************************************************************/
/*
*
*       Modified to test supervisor with the AV on the bench with the sonar information
*       simulated by using a memory map as input environment. This Supervisor implements the
*       CAN interface, and only uses the screen for monitor purposes.
*       Prepared by Enique M. Acuña, Oct. 1995
*
*/
/*************************************************************************
 INCLUDE FILES FOR SUPERVISOR MODULE
 *************************************************************************/
#include<alloc.h>
#include<math.h>
#include<stdlib.h>
#include<stdio.h>
#include<conio.h>
#include"ibmcom1.c"
#include"ZZ_CAN.H"
#include"ZZ_CARSP.H"
#include"ZZ_MISC.H"
#include"ZZ_MAP.H"
#include"ZZ_GRAPH.H"
#include"ZZ_SUPR2.H"
#include"ZZ_QUEUE.H"

#define SPEED 9600

#define FALSE 0
#define FIRST_TIME 2
#define TRUE 1
#define LEFTY 1
#define RIGHTY 0
#define TRESHOLDT 0.50 /* meters */
#define TRESHOLD  0.50 /* meters */
#define MAXTRIES 2
#define GET_OUT_WAIT 15
#define FUZZY 0
#define PLANNER 1
#define MAX_tries 4

#define SendToNav               1
```

```
#define Req_Nav_RS_C_SA 2
#define Req_Nav_XYpos 3
#define Req_Vis_Son1to4 4
#define Req_Vis_Son56  5
#define RS_C_Cd_PF              0xED /* 237 */
#define XY_pos_PF               0xEC /* 236 */
#define Son1to4_PF              0xEB /* 235 */
#define Son56_PF                0xE9 /* 233 */
#define H                                       18.88 /* distance from front to back axis inches */
#define Radius                  8.06 /* radius of back wheels in inches */

extern int graph,escape;

FILE *ofp,*psim;

double  ZZ_Max(double,double);
double  ZZ_Min(double,double);

extern void
        ZZ_SupertoMap(double *, double *,double *,
                                double *,double *,double *,double *,int *,int *,
                                double *,double *),
        ZZ_SupertoConsl(double *,double *, int *);

void CAN_interface(int);
void action(void);
void ascii(unsigned char);
unsigned char i_ascii(unsigned char, unsigned char);

/* variables for CAN interface */
unsigned char n_iascii,l_ascii,r_ascii;
static int mot_onoff=1;
static double Comp_dot;


static int line_of_file=0;
static double       WayX=1.0,WayY=4.0,i_WayX=1.0,i_WayY=4.0;
static double  TargetX,TargetY;
int NewTarget=FALSE,yes=FALSE;

/********  INPUT LINGUISTIC VARIABLE (MEMBERSHIP FUNCTION DEF.)  ******
 * ILV=[left straight right behind small med long]
 */
static float
        ILV[]={270,0,90,180,0,1,2,56.4,25.2,56.4,12.6,.315,.63,.63},

/* ILV2=[forward backward straight left right too_close very_close close]
*/
        ILV2[]={.25,-.25,0,-.25,.25,0,.2,0,.05,.05,25.2,.05,.05,.4,.7,.8},

/********* OUTPUT LINGUISTIC VARIABLES (MEMBERSHIP FUNCTION DEF.)  ******
 * OLV={hardleft straight hardright straight stop slow med]
 */
        OLV[]={-60,0,60,0,0,.7,1.5},
```

```
/* OLV2 = [stop reverse slow_down go_forward hardright hardleft to_right
                                    to_left hardright2 hardleft2 slow_a_bit]
 */
        OLV2[]={0,-1.6,-.9,-1.6,60,-60,60,-40,40,-60,-.7};
static  double
        CarXc[NUM_EDGES],       /**** ROBOT EDGES RELATIVE X-POSITION
                                        FROM REFERENCE POINT (pixels)****/

        CarYc[NUM_EDGES],       /**** ROBOT EDGES RELATIVE Y-POSITION
                                        FROM REFERENCE POINT (pixels)****/

        CarR[NUM_EDGES],        /**** ROBOT EDGES RELATIVE RADIUS
                                        FROM REFERENCE POINT (meters)****/

        CarAng[NUM_EDGES],      /**** ROBOT EDGES RELATIVE ANGLE
                                        FROM FRONT OF ROBOT (radians)****/

        Xc[NUM_SENSORS],        /**** SENSOR X-POSITIONS (pixels) ****/

        Yc[NUM_SENSORS],        /**** SENSOR Y-POSITIONS (pixels) ****/

        Rxy[NUM_SENSORS],       /**** SENSOR RADIUS FROM REFERENCE ****/

        AngleXY[NUM_SENSORS],   /**** SENSOR ANGLE FROM REF. POINT ****/

        Range[NUM_SENSORS],     /**** SONAR RANGE INFORMATION     ****/

        Compass=0.0,            /**** ROBOT BEARING               ****/

        Xposition,              /**** GLOBAL X-POSITION           ****/

        Yposition,              /**** GLOBAL Y-POSITION           ****/

        RoadSpeed=0.0,          /**** DESIRED ROBOT SPEED (m/sec) ****/

        SteeringAngle,          /**** DESIRED STEERING (radians) ****/

        ARoadSpeed,             /**** ACTUAL ROBOT SPEED (m/sec) ****/

        ASteeringAngle,         /**** ACTUAL STEERING (radians)  ****/

        WayPointDist,           /**** CURRENT WAYPOINT DISTANCE
                                        RELATIVE TO ROBOT (meters) ****/

        WayPointAngle,          /**** CURRENT WAYPOINT ANGLE RELATIVE
                                        TO FRONT OF ROBOT (radians) ****/

        ABrake,                 /**** ACTUAL BRAKE STATUS (ON/OFF) ****/

        Front,
        FrontR,
        Back,
        BackR,
        Right,
```

```
        RightR,
        Left,
        LeftR,
        Right2,
        RightR2,
        Left2,
        LeftR2,
        CurrentWaypointX=1.0,
        CurrentWaypointY=4.0,
        OLDXposition,
        OLDYposition,
        RefCompass;

static  int
        Xconvert,
        Yconvert,
        tries=0,
        NOT_REACHED,
        GET_OUT=OFF,
        GSTEER,
        METHOD;
static double timee = 0;

unsigned timer2=0,
         timer1=0;

ZZ_Point                Point;
QUEUE           PQueue;

void    ZZ_GetSensorData(void);
void    ZZ_ReachTarget(void);
void    ZZ_ReachWayPoint(void);
void    ZZ_CollisionAvoidance(void);
int ZZ_Planner(int x, int y,int tx,int ty,QUEUE *PATH);

/*************************************************************************
 *                                      *
 *      This Function must be called when starting      *
 *      program execution                       *
 *                                      *
 *************************************************************************/
void    ZZ_InitSuper(void)
{
        int i;

/* installing of serial communications drivers */
        init_port(2, SPEED,COM_NONE,1);

        Xposition=i_WayX;
        Yposition=i_WayY;
        ZZ_Real2Screen(Xposition,Yposition,&Xconvert,&Yconvert);
printf("XYpos %3.2f %3.2f XYcon %3.2f %3.2f ",
                Xposition,Yposition,Xconvert,Yconvert);
```

```
        for(i=0;i<NUM_EDGES;i++)
                ZZ_Cart2Polar(CarEdgeX[i],CarEdgeY[i],&CarAng[i],&CarR[i]);

        for(i=0;i<NUM_SENSORS;i++)
                ZZ_Cart2Polar(Xr[i],Yr[i],&AngleXY[i],&Rxy[i]);

        for(i=0;i<NUM_SENSORS;i++)
                ZZ_Polar2Cart(AngleXY[i]+Compass,
                Rxy[i]*SCREENCONVERT,&Xc[i],&Yc[i]);

        for(i=0;i<NUM_EDGES;i++)
                ZZ_Polar2Cart(CarAng[i]+Compass,CarR[i]*SCREENCONVERT,
                                                &CarXc[i],&CarYc[i]);


/*******
 ******* PASS MEMORY ADDRESSES OF VARIABLES USED BY SUPERVISOR SUB-MODULES
 *******/
        ZZ_SupertoMap(CarXc,CarYc,Xc,Yc,Range,&Compass,&RoadSpeed,&Xconvert,
                        &Yconvert,&Xposition,&Yposition);
//      ZZ_SupertoConsl(&WayX,&WayY,&NewTarget);

        clrscr();
        action();


//      ZZ_InitGraph();
        ZZ_InitMap();

        Front=Range[0];
        FrontR=0;
        Back=Range[1];
        BackR=0;
        Right=Range[3];
        RightR=0;
        Left = Range[2];
        LeftR=0;
        Right2 = Range[5]=0;
        RightR2 = 0;
        Left2 = Range[4];
        LeftR2 = 0;
//      ofp=fopen("output","w");
        psim=fopen("sim_gr.txt","w");
}

/*********************************************************************
 *                                                                   *
 *       MAIN SUPERVISOR LOOP                    *
 *-------------------------------------------------------------- *
 *                                                                   *
 * The supervisor tasks are divided by function names                *
 *                                                                   *
 *********************************************************************/
#define start 1
```

```
void    ZZ_SuperLoop(void)
{

        static int FLAG = start;

        if (escape) {
                        RoadSpeed=0.0;
                        SteeringAngle=0.0;
                        mot_onoff=0;
                        CAN_interface(SendToNav);
                        yes=FALSE;
                        exit(1);
                }


        ZZ_GetSensorData();

fprintf(psim,"%d %4.2f %4.2f %4.2f %4.2f %4.2f %4.2f \n",line_of_file,
                                Range[0],Range[1],Range[2],Range[3],Range[4],Range[5]);
fprintf(psim,"%4.2f %4.2f %4.2f %4.2f %4.2f \n",ARoadSpeed,Compass,
                                ASteeringAngle,Xposition,Yposition);

        if(--FLAG<0)
        {
                ZZ_UpdateMap();
                FLAG = start;
        }
        ZZ_ReachTarget();
        ZZ_ReachWayPoint();
        ZZ_CollisionAvoidance();
        {
                int secq;
                double minq;
                secq =60.0l*modf(timee/60.0l,&minq);
        gotoxy(1,1);printf("TIME : %4.0lf minutes, %2d seconds",minq,secq);

fprintf(psim,"%4.0lf %2d ",minq,secq);

        }

        timee += 0.2l;
//      ZZ_SendToNav();
        CAN_interface(SendToNav);
fprintf(psim,"%4.2f %4.2f \n",RoadSpeed,SteeringAngle);

                /*********    Handles Keyboard Presses  **********/
                keypress_handler();


line_of_file++;
}

/*****************************************************************
 *                                                               *
```

```
*   THIS FUNCTION OBTAINS SENSOR INFORMATION FROM          *
*      THE PROPULSION MODULE AND VISION MODULE             *
*      THROUGH THE CONTROLLER AREA NETWORK REQUEST COMMAND          *
* ---------------------------------------------------------*
*                                                                   *
*  Currently, The CAN is simulated in software. Functions      *
*   used to access the CAN are subject to change when the CAN is *
*   acually implemented                                          *
*                                                                   *
*********************************************************************/
void    ZZ_GetSensorData(void)
{
        int i=1;
        ZZ_CAN_Request(HIGHPRIORITY,SUPERVISOR,VISION,&DataContent,
                                                        Range,&DataNum);


/*      ZZ_CAN_Request(HIGHPRIORITY,SUPERVISOR,NAVIGATION,&DataContent,
                                                        Data,&DataNum);
        i=1;
        ARoadSpeed      = Data[i++];
        Compass         = Data[i++];
        Xposition       = Data[i++];
        Yposition       = Data[i++];
        ASteeringAngle  = Data[i++];
        ABrake          = Data[i++];
*/
        CAN_interface(Req_Nav_RS_C_SA);
        CAN_interface(Req_Nav_XYpos);
/*      CAN_interface(Req_Vis_Son1to4);
        CAN_interface(Req_Vis_Son56);
*/

        ZZ_Real2Screen(Xposition,Yposition,&Xconvert,&Yconvert);

        for(i=0;i<NUM_SENSORS;i++)
                ZZ_Polar2Cart(AngleXY[i]+Compass,
                        Rxy[i]*SCREENCONVERT,&Xc[i],&Yc[i]);

        for(i=0;i<NUM_EDGES;i++)
                ZZ_Polar2Cart(CarAng[i]+Compass,CarR[i]*SCREENCONVERT,
                                        &CarXc[i],&CarYc[i]);

}

/*********************************************************************
*   THIS FUNCTION DECIDES WHEN TO GENERATE A PATH AND          *
*   SELECTS THE CURRENT WAYPOINT TO BE USED BY THE             *
*   FUZZY CONTROLLERS. WHEN TARGET REACHED, IT WAITS UNTIL NEW   *
*   TARGET HAS BEEN SELECTED.                                    *
*********************************************************************/
void    ZZ_ReachTarget(void)
{
        int TX,TY;
```

```
/***
 *** If current position of car is close to current waypoint
 *** get new waypoint.
 ***/
if(fabs(CurrentWaypointX-Xposition)<TRESHOLD &&
  fabs(CurrentWaypointY-Yposition)<TRESHOLD)
{
        /***
         *** If there are other waypoints in path
         *** get the next point as set as the current
         *** waypoint .
         ***/
        if(!isempty(&PQueue))
        {
                dequeue(&PQueue,&Point);
                ZZ_Screen2Real(Point.x,Point.y,
                 &CurrentWaypointX,&CurrentWaypointY);
        }
        else
        /***
         *** If close to target do nothing.-- finish program
         ***/
        if(fabs(TargetX-CurrentWaypointX)<TRESHOLDT &&
                fabs(TargetY-CurrentWaypointY)<TRESHOLDT)
        {
                CurrentWaypointX = Xposition;
                CurrentWaypointY = Yposition;
                RoadSpeed=0.0;
                SteeringAngle=0.0;
                mot_onoff=0;
                CAN_interface(SendToNav);
                yes=FALSE;
                exit(1);
        }
        /***
         *** Otherwise, generate a new path
         *** and use the first point as the current
         *** waypoint.
         ***/
        else
                NewTarget = TRUE;
}

/***
 ***     New Path has been generated or new target has been selected
 ***/
if(NewTarget)
{
/*** Generate new path and set current waypoint as first point
 *** in new path
 ***/
        if(NewTarget == FIRST_TIME)
        {
```

```
                gotoxy(1,1);
                printf("TIME : %4.0lf minutes, %2d seconds",0.0l,0);
                TargetX = WayX;
                TargetY = WayY;
                timee = 0;
                timer1 = 0;
                METHOD = PLANNER;
                yes=FALSE;
        }
        ZZ_Real2Screen(TargetX,TargetY,&TX,&TY);
        if(!ZZ_Uncovered(TX,TY,4))
        {
                if(METHOD == FUZZY)
                        METHOD = PLANNER;
                else
                        METHOD = FUZZY;
        }

        if(METHOD == PLANNER)
        {
        resetqueue(&PQueue);
        NOT_REACHED = ZZ_Planner(Xconvert,Yconvert,TX,TY,&PQueue);
        dequeue(&PQueue,&Point);
        ZZ_Screen2Real(Point.x,Point.y,
                &CurrentWaypointX,&CurrentWaypointY);
        }
        else
        {
                CurrentWaypointX = TargetX;
                CurrentWaypointY = TargetY;
                NOT_REACHED = 0;
        }
        NewTarget = FALSE;
}
/***
 ***    If target is visible within sonar range, then
 ***    ignore path and go directly to target location.
 ***    This only applies when path was not found.
 ***/
if(NOT_REACHED)
{
        ZZ_Real2Screen(TargetX,TargetY,
                                &TX,&TY);
        if(ZZ_Uncovered(TX,TY,7))
        {
                CurrentWaypointX = TargetX;
                CurrentWaypointY = TargetY;
        }
}
/***
 ***    Change Waypoint coordinates from stationary cartesian
 ***    coordinates to polar coordinates relative to front of car
 ***/
WayPointAngle = ZZ_Atan2(CurrentWaypointX-Xposition,
```

```
                CurrentWaypointY-Yposition);
        WayPointAngle -= Compass;
        ZZ_LimitAngle(&WayPointAngle);
        WayPointDist = ZZ_Range(CurrentWaypointX-Xposition,
                        CurrentWaypointY-Yposition);

}

/*********************************************************************
*                   CONTROLLER # 1                  *
*       TRACK CURRENT WAYPOINT                      *
*       FUZZY INFERENCE RULES                   *
*********************************************************************/
void    ZZ_ReachWayPoint(void)
{
float A,B,SumAA=0.0,SumSA=0.0,SumB=0.0,mui,muj,mu;
int i,j;


/******* CONVERT Waypointangle from radians to degrees     ******/
        WayPointAngle *= 180.0/M_PI;

        if(WayPointAngle<0)
                WayPointAngle += 360.0;

        for(j=0;j<4;++j)
                {
                muj=(float)exp(-pow(((WayPointAngle-ILV[j])/ILV[7+j]),2));
                for(i=0;i<3;++i)
                        {
                        mui=(float)exp(-pow(((WayPointDist-ILV[4+i])/ILV[11+i]),2));
                        if(i == 2)
                                {
                                if(WayPointDist > ILV[6])
                                        mui=1;
                                }
                        mu=muj*mui;
                        SumAA +=mu*OLV[j];
                        SumB += mu;
                        if (j == 3)
                                SumSA += -mu*OLV[4+i];
                        else
                                SumSA += mu*OLV[4+i];
                        }
                }

/*********************************************************************
*********
*********       DEFUZZIFICATION OF OUTPUT VARIABLES USING LARSEN'S RULE
*********
*********************************************************************/
        if(SumB >= -.05 && SumB <= .05)
                SumB=1;
        SteeringAngle = (double)(SumAA/SumB)*M_PI/180.0;
```

```
        RoadSpeed = (double)SumSA/SumB;
                if(yes)
                {
                float J,A,S;
                if(WayPointAngle>180)
                        A=(WayPointAngle-360)*M_PI/180;
                else
                        A=WayPointAngle*M_PI/180;
                S=SteeringAngle;

                J=.5*(float)pow(((float)pow(A,2)+(float)pow(S,2)),2);
//              fprintf(ofp,"\n%f %f %f %f %f",A,S,J,TargetX,TargetY);
                }
}

/*********************************************************************
*   This Function handles communication requests from          *
*                 other Modules through                  *
*          Controller Area Network                *
*********************************************************************/
void ZZ_SuperServer(byte SourceAddress,int *DataContent,
                                        double *Data, byte *DataNum)
{
        *DataContent = NOTALLOWED;
}

/*********************************************************************
*                  CONTROLLER # 2                  *
*       COLLISION AVOIDANCE AND                *
*       GET OUT OF TIGHT SPOTS                      *
*                FUZZY INFERENCE RULES              *
*****************************FORWARD*****************************/
void    ZZ_CollisionAvoidance(void)
{
        int uSP[]={1,1,1,2,2,2,2,2,2,3,2,2},
                uST[]={0,5,4,6,7,6,6,7,7,0,8,9},i;
        double max,min,SteeringChange=0,SpeedChange=0;
        float muj,sumSA=0,sumB=0,sumSPA=0;
        float A,R;
        float mu[12],f,b,r,l;

        /***
         *** Clear fuzzy output variables
         ***/
        for (i=0;i<12;++i)
                mu[i]=0.0;
        FrontR = Range[0] - Front;
        BackR  = Range[1] - Back;
        LeftR  = Range[2] - Left;
        RightR = Range[3] - Right;
        LeftR2 = Range[4] - Left2;
        RightR2= Range[5] - Right2;
        min = Front = Range[0];
        Back  = Range[1];
```

```
        min = ZZ_Min(min,Back);
        Left  = Range[2];
        min = ZZ_Min(min,Left);
        Right = Range[3];
        min = ZZ_Max(min,Right);
        Left2 = Range[4];
        min = ZZ_Min(min,Left2);
        Right2 = Range[5];
        min = ZZ_Min(min,Right2);
        /***
         *** Fuzzy Rule to slow down vehicle in the vicinity of obstacles
         ***/
        muj=(float)exp(-pow(((min-ILV2[7])/ILV2[15]),2));
        sumB+=muj;
        sumSPA+=muj*OLV2[10];

        /***
         *** Change from radians to degrees
         ***/
        SteeringAngle *= 180.0/M_PI;

/***
 ***        GET OUT OF TIGHT SPOTS USING A HEURISTIC METHOD
 ***          IN COMBINATION WITH THE REVERSE FUZZY RULES TO
 ***              AVOID COLLISION
 ***/
        if(timer1++>GET_OUT_WAIT)
        {
                timer1 = 0;
                OLDXposition = Xposition;
                OLDYposition = Yposition;
        }

        if(GET_OUT == OFF &&
                (fabs(Xposition-OLDXposition)<.1 &&
           fabs(Yposition-OLDYposition)<.1 &&
           timer1==GET_OUT_WAIT &&
                (fabs(TargetX-Xposition)>TRESHOLDT ||
           fabs(TargetY-Yposition)>TRESHOLDT)
         )
        )
        {
                if(WayPointAngle >= 0.0)
                {
                        GSTEER = LEFTY;
                        RefCompass = Compass + M_PI*.6;
                }
                else
                {
                        GSTEER = RIGHTY;
                        RefCompass = Compass - M_PI*.6;
                }
                ZZ_LimitAngle(&RefCompass);
                GET_OUT = ON;
```

```
                if(++tries>MAX_tries && METHOD == FUZZY)
                {
                        tries = 0;
                        GET_OUT = OFF;
                        NewTarget = TRUE;
                }

        }

        if(GET_OUT == ON)
        {
                if(fabs(RefCompass - Compass) < 0.1
                  || fabs(RefCompass - Compass) > 2*M_PI-.1
                  || Back <.2 || Right2<.1 || Left2 <.1
                  || (((Left<.4 && Left>.3) || (Right<.4 && Right>.3))
                        && (fabs(RefCompass - Compass) < M_PI*.5 ||
                           fabs(RefCompass - Compass) > 1.5*M_PI)))
                {
                        timer1 = 0;
                        GET_OUT = OFF;
                }
                if(GSTEER == LEFTY)
                        SteeringAngle = -45.0;
                else
                        SteeringAngle = 45.0;
                RoadSpeed = -0.2;
/*              ZZ_AddMax(1,&STOP,&FSpeedChange);
                SpeedChange=0.0; */
        }

/***
 ***     FUZZY RULES USED TO AVOID COLLISIONS
 ***/
 if(GET_OUT == OFF)
 {
                A=SteeringAngle;
                R=RoadSpeed;
                if(R > ILV2[0])
                        f=1;
                else
                        f=(float)exp(-pow(((R-ILV2[0])/ILV2[8]),2));
                if(R < ILV2[1])
                        b=1;
                else
                        b=(float)exp(-pow(((R-ILV2[1])/ILV2[9]),2));
                if(A > ILV2[4])
                        r=1;
                else
                        r=(float)exp(-pow(((A-ILV2[4])/ILV2[12]),2));
                if(A < ILV2[3])
                        l=1;
                else
                        l=(float)exp(-pow(((A-ILV2[3])/ILV2[11]),2));
```

```
mu[0]=(float)exp(-pow(((A-ILV2[2])/ILV2[10]),2))*
                    f*
                    (float)exp(-pow(((Front-ILV2[5])/ILV2[13]),2));

mu[1]=l*
                    f*
                    (float)exp(-pow(((Front-ILV2[5])/ILV2[13]),2));

mu[2]=r*
                    f*
                    (float)exp(-pow(((Front-ILV2[5])/ILV2[13]),2));

mu[3]=l*
                    f*
                    (float)exp(-pow(((Left2-ILV2[5])/ILV2[13]),2));

mu[4]=r*
                    f*
                    (float)exp(-pow(((Right2-ILV2[5])/ILV2[13]),2));

mu[5]=(float)exp(-pow(((A-ILV2[2])/ILV2[10]),2))*
                    f*
                    (float)exp(-pow(((Right-ILV2[6])/ILV2[14]),2));

mu[6]=l*
                    f*
                    (float)exp(-pow(((Right-ILV2[6])/ILV2[14]),2));

mu[7]=r*
                    f*
                    (float)exp(-pow(((Left-ILV2[6])/ILV2[14]),2));

mu[8]=(float)exp(-pow(((A-ILV2[2])/ILV2[10]),2))*
                    f*
                    (float)exp(-pow(((Left-ILV2[6])/ILV2[14]),2));

mu[9]=(float)exp(-pow(((A-ILV2[2])/ILV2[10]),2))*
                    b*
                    (float)exp(-pow(((Back-ILV2[5])/ILV2[13]),2));

mu[10]=l*
                     b*
                     (float)exp(-pow(((Left2-ILV2[5])/ILV2[13]),2));

mu[11]=r*
                     b*
                     (float)exp(-pow(((Right2-ILV2[5])/ILV2[13]),2));

 for(i=0;i<12;++i)
         {
         sumSA+=mu[i]*OLV2[uST[i]];
         sumB+=mu[i];
         sumSPA+=mu[i]*OLV2[uSP[i]];
         }
```

```
        /*****************************************************************
         ***     DEFUZZIFY OUTPUTS                                       ***
         *****************************************************************/
        if(sumB >= -.05 && sumB <= .05)
                sumB=1;
        SteeringAngle*= M_PI/180.0;
        SteeringChange = (double)(sumSA/sumB)*M_PI/180.0;
        SpeedChange = (double)sumSPA/sumB;
        SteeringAngle += 2*SteeringChange;
        if(SteeringAngle<-45.0*M_PI/180.0)
                SteeringAngle = -45.0*M_PI/180.0;
        if(SteeringAngle>45.0*M_PI/180.0)
                SteeringAngle = 45.0*M_PI/180.0;
        RoadSpeed = RoadSpeed*(1+SpeedChange);
}
}

/*****************************************************************************
*       Returns Maximum Value Between Two Values          *
*****************************************************************************/
double  ZZ_Max(double value1, double value2)
{
        if(value1>value2)
                return(value1);
        return(value2);
}

/*****************************************************************
*       Information needed for Vision Simulation.       *
*       This function is only needed for simulation purposes    *
*****************************************************************/
void    ZZ_NavToVision(long *one,long *two,long *three)
{
        *one = &Xposition;
        *two = &Yposition;
        *three = &Compass;
}

/*****************************************************************************
*       Returns Minimum Value Between Two Values          *
*****************************************************************************/
double  ZZ_Min(double value1, double value2)
{
        if(value1<value2)
                return(value1);
        return(value2);
}

void action(void)
{
        int s;
        gotoxy(1,5);
        printf("TargetX:  ");
```

```
        do {
                gotoxy(20,5);
                s=scanf("%lf",&TargetX);
                flushall();
        } while(s==0);
        gotoxy(20,5);
        printf("%3.2f   ",TargetX);

        gotoxy(1,7);
        printf("TargetY:  ");
        do {
                gotoxy(20,7);
                s=scanf("%lf",&TargetY);
                flushall();
        } while(s==0);
        gotoxy(20,7);
        printf("%3.2f   ",TargetY);

        NewTarget = FIRST_TIME;
}


/*"PC-ENAT-CAN communications interface"*/

void CAN_interface(int choice)
{
        unsigned char bufTX[24];
        unsigned char bufRX[24];
        unsigned char buf[24];
        unsigned int temp;
        char i,j;
        bufTX[0]=0x09;  /* 0x09 (buf[0]) starts mess to TX */
        CLEAR_BUFFERS;
        switch(choice) {
        /* "1) Command D.Speed, D.S.Angle"*/
                case 1:
                        /* converts m/sec into RPM, 1 m/s = 46.53 RPM with
                                wheel radius of 8.08 inches */
                        temp=(int)(RoadSpeed*46.53+32767);
                        bufTX[1]=0x01;  /* indicates message #1 */
                        ascii(temp>>8);
                        bufTX[2]=l_ascii;
                        bufTX[3]=r_ascii;
                        ascii(temp);
                        bufTX[4]=l_ascii;
                        bufTX[5]=r_ascii;
                        temp=(int)(SteeringAngle*M_PI/180+32767);
                        ascii(temp>>8);
                        bufTX[6]=l_ascii;
                        bufTX[7]=r_ascii;
                        ascii(temp);
                        bufTX[8]=l_ascii;
                        bufTX[9]=r_ascii;
                        ascii(mot_onoff>>8);
```

```
                        bufTX[10]=l_ascii;
                        bufTX[11]=r_ascii;
                        ascii(mot_onoff);
                        bufTX[12]=l_ascii;
                        bufTX[13]=r_ascii;
                        send_command(bufTX);
                        wait_for_response(bufRX);
                        buf[0]=i_ascii(bufRX[2],bufRX[3]);
                        buf[1]=i_ascii(bufRX[4],bufRX[5]);
                        buf[2]=i_ascii(bufRX[6],bufRX[7]);
                        buf[3]=i_ascii(bufRX[8],bufRX[9]);
                        buf[4]=i_ascii(bufRX[10],bufRX[11]);
                        buf[5]=i_ascii(bufRX[12],bufRX[13]);
                        buf[6]=i_ascii(bufRX[14],bufRX[15]);
                        buf[7]=i_ascii(bufRX[16],bufRX[17]);
                        if (bufRX[1]==0x01) {  /* checks if correct mess is received */
/*                              gotoxy(1,16);
                                printf("Acknowledged: %x %x %x %x %x %x %x
%x",buf[0],buf[1],buf[2],buf[3],buf[4],buf[5],buf[6],buf[7]);
*/                              break;
                        }
                        else {
                                gotoxy(1,16);
                                printf("Not_Acknowledged");
                                break;
                                }
        /*"2) Request ARSpeed, Compass, Comp_d"*/
                case 2:
                        bufTX[1]=0x02;  /* indicates message #2 */
                        bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
                        bufTX[3]=0x30;
                        bufTX[4]=0x45;  /* bufTX[4] and [5] send RS_C_Cd_PF (0xED) */
                        bufTX[5]=0x44;  /* in ascii */
                        bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
                        bufTX[7]=0x30;
/*                      gotoxy(1,18);
                        printf("RS_C_Cd requested   ");
*/                      send_command(bufTX);
                        wait_for_response(bufRX);
                        buf[0]=i_ascii(bufRX[2],bufRX[3]);
                        buf[1]=i_ascii(bufRX[4],bufRX[5]);
                        buf[2]=i_ascii(bufRX[6],bufRX[7]);
                        buf[3]=i_ascii(bufRX[8],bufRX[9]);
                        buf[4]=i_ascii(bufRX[10],bufRX[11]);
                        buf[5]=i_ascii(bufRX[12],bufRX[13]);
                        if (bufRX[1]==0x02) {
                                temp=(buf[0]<<8) + buf[1];
                                ARoadSpeed=(temp-32767.0)/10;
/*                              printf("buf[0:1] %x %x buf[2,3] %x %x ",buf[0],buf[1],buf[2],buf[3]);
                                printf("buf[4,5] %x %x ",buf[4],buf[5]);
*/      temp=(buf[2]<<8) + buf[3];
                                Compass+=(temp-32767.0);
                                temp=(buf[4]<<8) + buf[5];
                                Comp_dot=(temp-32767.0)/10;
```

```
                                        ASteeringAngle=ZZ_Atan2((Comp_dot*H),(ARoadSpeed*Radius));
                                        /* converts RPM into m/s */
                                        ARoadSpeed*=0.0215;
                                        /* converts degrees into radians */
                                        Compass*=M_PI/180;
/*                                      gotoxy(1,19);
                                        printf("Road Speed: %5.2f",ARoadSpeed);
                                        gotoxy(1,20);
                                        printf("Compass:   %5.2f",Compass);
                                        gotoxy(1,21);
                                        printf("AS.Angle:  %5.2f ",ASteeringAngle);
*/                                      break;
                                        }
                                else {
                                        gotoxy(1,19);
                                        printf("No_expected_response");
                                        break;
                                        }
        /*"3) Request XY_pos"*/
                        case 3:
                                bufTX[1]=0x03; /* indicates message #3 */
                                bufTX[2]=0x30; /* bufTX[2] and [3] send 0x00 in ascii */
                                bufTX[3]=0x30;
                                bufTX[4]=0x45; /* bufTX[4] and [5] send XY_pos_PF (0xEC) */
                                bufTX[5]=0x43; /* in ascii */
                                bufTX[6]=0x30; /* bufTX[6] and [7] send 0x00 in ascii */
                                bufTX[7]=0x30;
/*                              gotoxy(1,23);
                                printf("XY_pos requested  ");
*/                              send_command(bufTX);
                                wait_for_response(bufRX);
                                buf[0]=i_ascii(bufRX[2],bufRX[3]);
                                buf[1]=i_ascii(bufRX[4],bufRX[5]);
                                buf[2]=i_ascii(bufRX[6],bufRX[7]);
                                buf[3]=i_ascii(bufRX[8],bufRX[9]);
                                if (bufRX[1]==0x03) {
                                        /* convert inches into meters */
                                        Xposition+=0.0254*(((buf[0]<<8) + buf[1])-32767);
                                        Yposition+=0.0254*(((buf[2]<<8) + buf[3])-32767);
/*                                      gotoxy(1,24);
                                        printf("X-Y position %5.2f %5.2f ",Xposition,Yposition);
*/                                      break;
                                        }
                                else {
                                        gotoxy(1,24);
                                        printf("No_expected_response");
                                        break;
                                        }
        /*"4) Request Sonar1 to 4"*/
                        case 4:
                                bufTX[1]=0x04; /* indicates message #4 */
                                bufTX[2]=0x30; /* bufTX[2] and [3] send 0x00 in ascii */
                                bufTX[3]=0x30;
                                bufTX[4]=0x45; /* bufTX[4] and [5] send Son1to4_PF (0xEB) */
```

```
                        bufTX[5]=0x42;  /* in ascii */
                        bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
                        bufTX[7]=0x30;
/*                      gotoxy(35,13);
                        printf("Son1 to 4 requested  ");
*/                      send_command(bufTX);
                        wait_for_response(bufRX);
                        buf[0]=i_ascii(bufRX[2],bufRX[3]);
                        buf[1]=i_ascii(bufRX[4],bufRX[5]);
                        buf[2]=i_ascii(bufRX[6],bufRX[7]);
                        buf[3]=i_ascii(bufRX[8],bufRX[9]);
                        buf[4]=i_ascii(bufRX[10],bufRX[11]);
                        buf[5]=i_ascii(bufRX[12],bufRX[13]);
                        buf[6]=i_ascii(bufRX[14],bufRX[15]);
                        buf[7]=i_ascii(bufRX[16],bufRX[17]);
                        if (bufRX[1]==0x04) {
                                Range[0]=((buf[0]<<8) + buf[1])/100.0;
                                Range[1]=((buf[2]<<8) + buf[3])/100.0;
                                Range[2]=((buf[4]<<8) + buf[5])/100.0;
                                Range[3]=((buf[6]<<8) + buf[7])/100.0;
/*                              gotoxy(35,14);
                                printf("Sonar 1-2: %d  %d",Range[0],Range[1]);
                                gotoxy(35,15);
                                printf("Sonar 3-4: %d  %d",Range[2],Range[3]);
*/                              break;
                                }
                        else {
                                gotoxy(35,14);
                                printf("No_expected_response");
                                break;
                                }
        /*"5) Request Sonar5 6"*/
                case 5:
                        bufTX[1]=0x05;  /* indicates message #5 */
                        bufTX[2]=0x30;  /* bufTX[2] and [3] send 0x00 in ascii */
                        bufTX[3]=0x30;
                        bufTX[4]=0x45;  /* bufTX[4] and [5] send Son56_PF (0xE9) */
                        bufTX[5]=0x39;  /* in ascii */
                        bufTX[6]=0x30;  /* bufTX[6] and [7] send 0x00 in ascii */
                        bufTX[7]=0x30;
/*                      gotoxy(35,17);
                        printf("Sonar 5-6 requested  ");
*/                      send_command(bufTX);
                        wait_for_response(bufRX);
                        buf[0]=i_ascii(bufRX[2],bufRX[3]);
                        buf[1]=i_ascii(bufRX[4],bufRX[5]);
                        buf[2]=i_ascii(bufRX[6],bufRX[7]);
                        buf[3]=i_ascii(bufRX[8],bufRX[9]);
                        if (bufRX[1]==0x05) {
                                Range[4]=((buf[0]<<8) + buf[1])/100.0;
                                Range[5]=((buf[2]<<8) + buf[3])/100.0;
/*                              gotoxy(35,18);
                                printf("Sonar 5-6: %d  %d",Range[4],Range[5]);
*/                              break;
```

```
                        }
                else {
                        gotoxy(35,18);
                        printf("No_expected_response");
                        break;
                        }
        default:
                gotoxy(1,19);
                printf("Error");
                break;
        }
}

void ascii(n_iascii)
unsigned char n_iascii;
{
l_ascii=((n_iascii>>4) & 0x0F) + 0x30;
r_ascii=(n_iascii & 0x0F) + 0x30;
if (l_ascii>0x39)
        l_ascii+=0x07;
if (r_ascii>0x39)
        r_ascii+=0x07;
}

unsigned char i_ascii(ln,rn)
unsigned char ln,rn;
{
if (ln>0x39)
        ln-=7;
if (rn>0x39)
        rn-=7;
n_iascii=((ln-0x30) << 4) + (rn-0x30);
return n_iascii;
}
```

```
/*************************************************************************
 FILE            : ZZ_MAP2.C

 DESCRIPTION : ADAPTIVE MAPPING FILE FOR AUTONOMOUS VEHICLE

                 This file contains code for the ADAPTIVE MAPPING,
                 and functions used for simulating environment.

                 NOTE: When implementing the adaaptive mapping,
                       only one color needs to be verified, since
                       only a binary map is needed. The other colors
                       are used only during simulation.
-------------------------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE     : MARCH 22, 1993
*************************************************************************
Updates after 22 March1993 by Robert L. Shanley III

 LAST UPDATE : 18March94,22March94,23March94,28March94,12July94

*************************************************************************/
/*
*
*       This file contains modifications and functions to be used for testing the AV on
*       the bench with simulated input environment from a memory map
*       prepared by: Enrique M. Acuña, Oct. 1995
*       file name: ZZ_MAPM.C
*
*
*************************************************************************/


/*************************************************************************
 INCLUDE FILES FOR MAPPING FILE : SUPERVISOR MODULE
 *************************************************************************/
//#include<graphics.h>
#include<time.h>
#include<stdlib.h>
#include<math.h>
#include <conio.h>
#include"ZZ_CAN.H"
#include"ZZ_SUPR2.H"
#include"ZZ_CARSP.H"
#include"ZZ_MAP.H"

#define M_PI_344.712388981

extern  int
        graph;

static  double
        *CarXc,
```

```
        *CarYc,
        *Xc,        /**** X SENSOR POSITIONS (pixels) ****/
        *Yc,        /**** Y SENSOR POSITIONS (pixels) ****/
        *Range,
        *Compass,
        *RoadSpeed,
        *Xposition,
        *Yposition;


static  int
        *Xconvert,
        *Yconvert,
        Crash;

int s_byte=sizeof(unsigned int)*8; /* the size of a byte in bits */
unsigned int map[480][23],ones;  /* map[row][column] -- the actual map*/
unsigned int room[480][23];  /* room in memory as input to program */
double theta=.17444;   /* arc of sensor in radians (10 deg) */
bit zero={0,480};   /* (x,y) origin in the bit field*/
float resolution=0.0250;      /* map resolution in m/bit */
float max_dist=0.250;      /*maximum distance of sonar in meters */
int maxx,maxy;        /* screen graphic variables (screen size)*/


/*****************************************************************
 *                                                               *
 *       GET MAIN SUPERVISOR PRIVATE VARIABLE ADDRESSES USED BY  *
 *       MAPPING FUNCTIONS AND ASSIGN THEM TO LOCAL PRIVATE      *
 *       VARIABLES.                                              *
 *                                                               *
 *****************************************************************/
void    ZZ_SupertoMap(double *one, double *two,double *three,
                      double *four,double *five,double *six,
                      double *seven,int *eight,int *nine,double *ten,
                      double *eleven)
{
        CarXc = one;
        CarYc = two;
        Xc = three;
        Yc = four;
        Range = five;
        Compass = six;
        RoadSpeed = seven;
        Xconvert = eight;
        Yconvert = nine;
        Xposition=ten;
        Yposition=eleven;
}



/*****************************************************************
```

```
 *                                                                  *
 *        INITIALIZE MAP TO ALL OCCUPIED SPACE.                     *
 * ----------------------------------------------------------------*
 *        NOTE: ZZ_DrawRoom is used only for simulation purposes.   *
 *            ZZ_DrawCar is not necessary, it is used to locate     *
 *            the robot on the screen.                              *
 *                                                                  *
 ********************************************************************/
void    ZZ_InitMap(void)
{
        int row,column,i;

/* clear the bit field (i.e. fill all bits with '1')
 * a '1' in the bit field represents occupied space
 * while a '0' in the bit field is unoccupied space
 */
        for(i=0;i<s_byte;++i)
                ones += pow(2,i);


        maxx=640; /* maximum values for x and y coordinates for screen */
        maxy=480;

        for(row=0;row<=479;++row)
                for(column=0;column<=22;++column)
                        map[row][column]=ones;

        ZZ_DoRoom();
}



/********************************************************************
 *                                                                  *
 *        ADAPTIVE MAPPING DONE HERE.                               *
 *-----------------------------------------------------------------*
 *        Note the use of different colors. Again this only         *
 *        applies to simulation. When implemented, only one color   *
 *        should be used.                                           *
 *                                                                  *
 ********************************************************************/
void    ZZ_UpdateMap(void)
{
        int row,column,r,i,ii;
        double x=0.0,y=0.0,d=0.0,phi=0.0;
        double w,a,b,p,q,m1,m2,m3,t1,t2,t3,tmp_y,tmp_x;
        bit l_bit,r_bit,tmp_bit,top_bit,botm_bit;

        for(ii=0;ii<NUM_SENSORS;ii++)
        {

/* r is the flag which represents the maximum
 * distance of the sensor.  r=0 places '1's at
 * distance d which signifies an obstacle.  Works
```

```
* even if that space has been previously cleared
* to signify a moving obstacle.  r=1 means the
* sensor was maxed out and don't place a '1'
* at distance d.
*/

                x = Xc[ii] + *Xconvert;     /*Current location*/
                y = Yc[ii] + *Yconvert;

                ZZ_Screen2Real((int)x,(int)y,&x,&y);
                d=Range[ii];                /*sonar distance*/
                r=0;
                if (d >= max_dist)          /*check dist for max sonar dist.*/
                        r=1;
                phi=*Compass+NominalAngle[ii];      /*heading angle*/

/* Make sure that the heading angle of the sensor
 * is in the correct clockwise format between
 * 0 and 2pi. Starting at this point, all code
 * must be included in the robot simulation.
 */
                        if (phi >= -0.01 && phi <=0.01)
                                phi=0;
                        else if (phi < 0.0)
                                phi=2*M_PI+phi;
                        else if (phi > 2*M_PI)
                                phi=phi-2*M_PI;

                w=d/cos(theta);   /* w is length of triangle sides */

                        a=x+w*sin(phi+theta);     /*find the two other triangle locations*/
                        b=y+w*cos(phi+theta);
                        p=x+w*sin(phi-theta);
                        q=y+w*cos(phi-theta);
                        if(p-a > -.05 && p-a < .05)
                                p=a;

                        m1=(b-y)/(a-x);           /*find slopes of triangle lines*/
                        if(m1 > -0.01 && m1 < 0.01)
                                m1=0.0;
                        m2=(q-y)/(p-x);
                        if(m2 > -0.01 && m2 < 0.01)
                                m2=0.0;
                        if(p==a)
                                m3=10e10;
                        else
                                m3=(q-b)/(p-a);
                        if(m3 > -0.01 && m3 < 0.01)
                                m3=0.0;
                        t1=y-x*m1;          /*find y-intercept of triangle lines*/
                        t2=y-x*m2;
                        t3=q-p*m3;
```

```
/* Because the output of the sonar is a cone, each
 * case in the cartesian plane must be accounted for.
 * The following clears the cones in the top half
 * of the plane.
 */
                if (a > p)
                        {

/* The following are when the cone has one
 * side horizontal or slope of zero.
 */
                        if (y == b)
                                {
                                l_bit=xy_to_bit(x,y);  /* Coordinates of left and  */
                                r_bit=xy_to_bit(a,b);  /* right most bits for that */
                                clear_row(l_bit,r_bit,1,r);   /* row */
                                }
                        else if (y == q)
                                {
                                l_bit=xy_to_bit(p,q);
                                r_bit=xy_to_bit(x,y);
                                clear_row(l_bit,r_bit,2,r);
                                }
                        tmp_bit=xy_to_bit(x,y);  /* generate the incremental bit */
                        tmp_bit.row_bit-=1;      /* increment the incremental bit */
                        if (b >= q)      /* determine the top bit of the triangle*/
                                top_bit=xy_to_bit(a,b);
                        else
                                top_bit=xy_to_bit(p,q);
                        tmp_y=y;       /*the line of bits currently correcting */
                        tmp_x=x;

/* The map is defined by bits in the following way:
 *    row  0 1 2 3 4 ...
 *  column
 *                  0
 *                  1
 *                  2
 * and so forth.  This means to clear a triangle from bottom
 * to top, the top bit has a lower value than the bottom bit.
 */
                        while(top_bit.row_bit <= tmp_bit.row_bit)
                                {

/* t is the flag which determines which side to place
 * the '1's when an obstical is pressent. t=0 signifies
 * no '1's or reset, t=1 signifies '1's on the right side,
 * and t=2 signifies '1's on the left side of the triangle.
 */
                                int t=0;
                                tmp_y+=resolution;   /*increment to next line of bits */

/* Determine the xy location of the right bit
```

```
 * and convert it to a bit map location
 */
                                        if (tmp_y > b && m3 != 0)
                                                {
                                                t=1;
                                                tmp_x=(tmp_y-t3)/m3;
                                                }
                                        else if (m1 == 0.0)
                                                tmp_x=tmp_x+resolution;
                                        else
                                                tmp_x=(tmp_y-t1)/m1;
                                        r_bit=xy_to_bit(tmp_x,tmp_y);

/* Determine the xy location of the left bit
 * and convert it to a bit map location.
 */
                                        if (tmp_y >q && m3 != 0)
                                                {
                                                t=2;
                                                tmp_x=(tmp_y-t3)/m3;
                                                }
                                        else if (m2 == 0.0)
                                                tmp_x=tmp_x-resolution;
                                        else
                                                tmp_x=(tmp_y-t2)/m2;
                                        l_bit=xy_to_bit(tmp_x,tmp_y);
/* clear that row of bits
 */
                                        clear_row(l_bit,r_bit,t,r);
/* Increment the temporary bit to check posistion
 * comparred to the top bit
 */
                                        tmp_bit=xy_to_bit(tmp_x,tmp_y+resolution);
                                        }

/* place '1's along top row of triangle if
 * the heading angle is 0 rad.
 */
                                if (m3 == 0 && r == 0)
                                        {
                                        int d,e,f,i;
                                        l_bit=xy_to_bit(p,q);
                                        r_bit=xy_to_bit(a,b);
                                        d=(int)floor((l_bit.col_bit)/s_byte);
                                        e=l_bit.row_bit;
                                        f=(int)floor((r_bit.col_bit)/s_byte);
                                        for (i=d;i<=f;++i)
                                                map[e][i]=ones;
                                        }
                                }
/* This section is the same as the above except
 * we are now concerned with changing the bits
 * if the triangle is in the bottom plane.  The
 * difference between the two are sign and line
```

```
* changes.
*/
                else if (p > a)
                        {

                        if (y == b)
                                {
                                l_bit=xy_to_bit(x,y);
                                r_bit=xy_to_bit(a,b);
                                clear_row(l_bit,r_bit,1,r);
                                }
                        else if (y == q)
                                {
                                l_bit=xy_to_bit(p,q);
                                r_bit=xy_to_bit(x,y);
                                clear_row(l_bit,r_bit,2,r);
                                }
                        tmp_bit=xy_to_bit(x,y);
                        tmp_bit.row_bit+=1;
                        if (b <= q)
                                botm_bit=xy_to_bit(a,b);
                        else
                                botm_bit=xy_to_bit(p,q);
                        tmp_y=y;
                        tmp_x=x;
                        while(botm_bit.row_bit >= tmp_bit.row_bit)
                                {
                                int t=0;
                                tmp_y-=resolution;
                                if (tmp_y < q && m3 != 0)
                                        {
                                        tmp_x=(tmp_y-t3)/m3;
                                        }
                                else if (m2 == 0.0)
                                        tmp_x=tmp_x+resolution;
                                else
                                        tmp_x=(tmp_y-t2)/m2;
                                r_bit=xy_to_bit(tmp_x,tmp_y);
                                if (tmp_y < b && m3 != 0)
                                        {
                                        tmp_x=(tmp_y-t3)/m3;
                                        }
                                else if (m1 == 0.0)
                                        tmp_x=tmp_x-resolution;
                                else
                                        tmp_x=(tmp_y-t1)/m1;
                                l_bit=xy_to_bit(tmp_x,tmp_y);
                                clear_row(l_bit,r_bit,t,r);
                                tmp_bit=xy_to_bit(tmp_x,tmp_y-resolution);
                                }

                        }

/* This section of code clears triangles which
```

```
* are split by the horizontal plane.
*/
                else if ((phi > M_PI_2-theta && phi < M_PI_2+theta)||
                                        (phi > M_PI_34-theta && phi < M_PI_34+theta))
                        {


                        tmp_bit=xy_to_bit(x,y);  /* generate the incremental bit */
                        if (b >= q)             /* Find top and botm corners */
                                {               /* of the triangle */
                                top_bit=xy_to_bit(a,b);
                                botm_bit=xy_to_bit(p,q);
                                }
                        else
                                {
                                top_bit=xy_to_bit(p,q);
                                botm_bit=xy_to_bit(a,b);
                                }

/* This block of code clears the horizontal
 * triangle in the right (positive x) plane.
 * Clearing horizontal triangles means clearing
 * the top half of the triangle first and then
 * going back and clearing the bottom half of
 * the triangle.
 */

                        if (a > x)
                                {

                                tmp_y=y; /*the line of bits currently correcting */

/* Clear the top half of the triangle.
 */
                                while(top_bit.row_bit <= tmp_bit.row_bit)
                                        {
                                        int t=1;
                                        l_bit=tmp_bit; /*determine the left most bit*/
                                        tmp_x=(tmp_y-t3)/m3;
                                        r_bit=xy_to_bit(tmp_x,tmp_y);/*the right most bit*/
                                        clear_row(l_bit,r_bit,t,r);  /*clear the row*/
                                        tmp_y+=resolution;   /*increment to next line of bits */

/* The following block of code determines
 * the orrientation of the triangle and
 * specifies the line to which the next
 * row originates.
 */
                                        if (phi < M_PI_2)
                                                tmp_x=(tmp_y-t2)/m2;
                                        else
                                                {
                                                if (m2 == 0)
                                                tmp_x=tmp_x+resolution;
```

```
                                        else
                                                tmp_x=(tmp_y-t2)/m2;
                                        }
                                tmp_bit=xy_to_bit(tmp_x,tmp_y);
                                }

/* GO BACK TO HORIZONTAL AXIS */
                        tmp_y=y-resolution;
                        if (phi < M_PI_2)
                                {
                                if (m1 == 0)
                                        tmp_x=tmp_x+resolution;
                                else
                                        tmp_x=(tmp_y-t1)/m1;
                                }
                        else
                                        tmp_x=(tmp_y-t1)/m1;

/* Repeat the above for the bottom half of the triangle
 */
                        tmp_bit=xy_to_bit(tmp_x,tmp_y);
                        while(botm_bit.row_bit >= tmp_bit.row_bit)
                                {
                                int t=1;
                                l_bit=tmp_bit;
                                tmp_x=(tmp_y-t3)/m3;
                                r_bit=xy_to_bit(tmp_x,tmp_y);
                                clear_row(l_bit,r_bit,t,r);
                                tmp_y-=resolution;
                                if (phi < M_PI_2)
                                        {
                                        if (m1 == 0)
                                                tmp_x=tmp_x+resolution;
                                        else
                                                tmp_x=(tmp_y-t1)/m1;
                                        }
                                else
                                                tmp_x=(tmp_y-t1)/m1;

                                tmp_bit=xy_to_bit(tmp_x,tmp_y);
                                }
                        }


/* This block of code is for the cases when the
 * horizontal triangle is in the negative x
 * direction.
 */
                if (a < x)
                        {

                        tmp_y=y;
                        while(top_bit.row_bit <= tmp_bit.row_bit)
                                {
```

```
                        int t=2;
                        r_bit=tmp_bit;
                        tmp_x=(tmp_y-t3)/m3;
                        l_bit=xy_to_bit(tmp_x,tmp_y);
                        clear_row(l_bit,r_bit,t,r);
                        tmp_y+=resolution;
                        if (phi < M_PI_34)
                                {
                                if (m1 == 0)
                                        tmp_x=tmp_x+resolution;
                                else
                                        tmp_x=(tmp_y-t1)/m1;
                                }
                        else
                                        tmp_x=(tmp_y-t1)/m1;
                        tmp_bit=xy_to_bit(tmp_x,tmp_y);
                        }
                tmp_y=y-resolution;
                if (phi > M_PI_34)
                        {
                        if (m2 == 0)
                                tmp_x=tmp_x+resolution;
                        else
                                tmp_x=(tmp_y-t2)/m2;
                        }
                else
                                tmp_x=(tmp_y-t2)/m2;


                tmp_bit=xy_to_bit(tmp_x,tmp_y);
                while(botm_bit.row_bit >= tmp_bit.row_bit)
                        {
                        int t=2;
                        r_bit=tmp_bit;
                        tmp_x=(tmp_y-t3)/m3;
                        l_bit=xy_to_bit(tmp_x,tmp_y);
                        clear_row(l_bit,r_bit,t,r);
                        tmp_y-=resolution;
                        if (phi > M_PI_34)
                                {
                                if (m2 == 0)
                                        tmp_x=tmp_x+resolution;
                                else
                                        tmp_x=(tmp_y-t2)/m2;
                                }
                        else
                                        tmp_x=(tmp_y-t2)/m2;
                        tmp_bit=xy_to_bit(tmp_x,tmp_y);
                        tmp_bit=xy_to_bit(tmp_x,tmp_y);
                        }
                }
        }
}
```

```
/*****************************************************************************
 * converts from x,y coordinates to bit coordinates                          *
 * x and y must both be in inches                                            *
 *****************************************************************************/

bit xy_to_bit(float x, float y)
        {
        double f,i;
        int a,b;
        bit tmp;

        f=modf((double)(x/resolution),&i);  /* Find the column bit */
        if(f >= 0 && f < 0.5)               /* because of the resolution, the*/
                a=zero.col_bit + (int)i;        /* rounding becomes a significant*/
        else if (f > 0 && f >= 0.5)         /* issue */
                a=zero.col_bit + (int)i + 1;
        else if (f < 0 && f > -0.5)
                a=zero.col_bit + (int)i;
        else
                a=zero.col_bit + (int)i - 1;

        f=modf((double)(y/resolution),&i);  /* find row bit */
        if(f >= 0 && f < 0.5)
                b=zero.row_bit - (int)i;
        else if (f > 0 && f >= 0.5)
                b=zero.row_bit - (int)i - 1;
        else if (f < 0 && f > -0.5)
                b=zero.row_bit - (int)i;
        else
                b=zero.row_bit - (int)i + 1;

        tmp.col_bit=a;
        tmp.row_bit=b;
        return(tmp);
        }


/*****************************************************************************
 * Clears one row of bits from the map i.e. make  every bit in the row 0     *
 *****************************************************************************/

 void clear_row(bit l_bit,bit r_bit,int t,int r)
        {
        int a,b,c,i,k=0;
        unsigned int j=ones;

/* a is an integer which signifies which column array element in the
 * map has the column bit for the left bit.
 */
        a=(int)floor((l_bit.col_bit)/s_byte);
```

```
/* b is an integer which signifies which row array element in the
 * map has the row bits.
 */

        b=l_bit.row_bit;

/* a is an integer which signifies which column array element in the
 * map has the column bit for the right bit.
 */

        c=(int)floor((r_bit.col_bit)/s_byte);

/* if an entire integer (8 consecutive bits) needs to be
 * cleared--go through loop.
 */

        for(i=a+1;i < c; ++i)
                map[b][i]=0;

/* If single bits of an integer need to be cleared */

/* when the row to be cleared is intirely within one integer */
        if (a == c)
                {
                for(i=l_bit.col_bit-(a*s_byte);i<=r_bit.col_bit-(c*s_byte);++i)
                        j=j-(int)pow(2,(s_byte-1-i));
/* if there is an obstacal to the right and we are not at
 * maximum sensor distance, place a '1' at the right edge.
 */
                if (t == 1 && r == 0)
                        k=(int)pow(2,(c*s_byte+s_byte-1)-r_bit.col_bit);
/* if there is an obstacal to the left and we are not at
 * maximum sensor distance, place a '1' at the left edge.
 */
                else if (t == 2 && r == 0)
                        k=(int)pow(2,(a*s_byte+s_byte-1)-l_bit.col_bit);
                map[b][a]=(map[b][a] & j) | k;
                return;
                }

/* when the row to clear is several integers.*/
        else
                {
                int i,j=ones;
/* clear left bit */
                for (i=l_bit.col_bit-(a*s_byte);i<=(s_byte-1);++i)
                        j=j-(int)pow(2,(s_byte-1-i));
                if (t == 2 && r == 0)
                        k=(int)pow(2,(a*s_byte+s_byte-1)-l_bit.col_bit);

                map[b][a]=(map[b][a] & j) | k;

                j=ones;
/* clear right bit */
```

```
                for (i=0;i<=r_bit.col_bit-(c*s_byte);++i)
                        j=j-(int)pow(2,(s_byte-1-i));
                if (t == 1 && r == 0)
                        k=(int)pow(2,(c*s_byte+s_byte-1)-r_bit.col_bit);

                map[b][c]=(map[b][c] & j) | k;
                }
        return;

        }

/*************************************************************************
 *              Getbit will return the value of the bit x,y (either '0' or '1' *
 *************************************************************************/

int getbit(int x,int y)
        {
        int a,b,tmp,mask=1;

        a=(int)floor(x/s_byte);         /* column integer holding x bit */
        b=y;                                                    /* row integer holding y bit */
        tmp=map[b][a] >> (int)(s_byte-1-(x-a*s_byte));  /* shift bit to bit #0*/
        if(mask & tmp)
                return(1);
        else
                return(0);
        }

int     ZZ_Uncovered(int TX, int TY,int res)
{
        int sum=0;

        sum += getbit(TX+res,TY+res);
        sum += getbit(TX+res,TY-res);
        sum += getbit(TX-res,TY+res);
        sum += getbit(TX-res,TY-res);
        if(sum > 3)
                return(0);
        return(1);
}

/*************************************************************************
*
*       this function is similar to getbit, but works with the input environment stored in
*       "room"
*
*************************************************************************/

int getbit2(int x,int y)
        {
        int a,b,tmp,mask=1;

        a=(int)floor(x/s_byte);         /* column integer holding x bit */
        b=y;                                                    /* row integer holding y bit */
```

```
        tmp=room[b][a] >> (int)(s_byte-1-(x-a*s_byte));  /* shift bit to bit #0*/
//      printf("x,y a,b room[b][a] %d %d %d %d %x ",x,y,a,b,room[b][a]);
        if(mask & tmp)
                return(1);
        else
                return(0);
        }

/*************************************************************************
*
*       This function generates an environment map in memory that is used as the
*       input for the simulated vision module test of the AV on the bench
*
*************************************************************************/
void ZZ_DoRoom(void)
{
int i,j;
/* room is full of zeros */
for (i=0;i<480;i++)
        for (j=0;j<23;j++)
                room[i][j]=0x0000;
for (i=25;i<35;i++)  /* partial top edge of room */
        for (j=0;j<21;j++)
                room[i][j]=0xFFFF;
for (i=469;i<479;i++)  /* partial bottom edge of room */
        for (j=0;j<21;j++)
                room[i][j]=0xFFFF;
for (i=36;i<470;i++)   /* left edge of room */
        room[i][0]=0xFFC0;
/* obstacle 1 */
for (i=100;i<=155;i++)
        for (j=5;j<8;j++)
                room[i][j]=0xFFFF;
for (i=100;i<=155;i++)
                room[i][4]=0x000F;
for (i=100;i<=155;i++)
                room[i][8]=0xE000;

/* obstacle 2 */
for (i=260;i<=315;i++)
        for (j=6;j<=7;j++)
                room[i][j]=0xFFFF;
for (i=260;i<=315;i++)
                room[i][5]=0x03FF;
for (i=260;i<=315;i++)
                room[i][8]=0xFFFC;

/* obstacle 3 */
for (i=200;i<=255;i++)
        for (j=10;j<=11;j++)
                room[i][j]=0xFFFF;
for (i=200;i<=255;i++)
                room[i][9]=0x3FFF;
for (i=200;i<=255;i++)
```

```
                room[i][12]=0xFFC0;

/* obstacle 4 */
for (i=330;i<=378;i++)
        for (j=10;j<=11;j++)
                room[i][j]=0xFFFF;
for (i=330;i<=378;i++)
                room[i][9]=0x3FFF;
for (i=330;i<=378;i++)
                room[i][12]=0xFFC0;

/* obstacle 5 */
for (i=379;i<=469;i++)
        for (j=9;j<=11;j++)
                room[i][j]=0xFFFF;
for (i=379;i<=469;i++)
                room[i][8]=0x1FFF;
for (i=379;i<=469;i++)
                room[i][12]=0xFFC0;

/* obstacle 6 */
for (i=85;i<=419;i++)
                room[i][16]=0xFFFE;
for (i=85;i<=419;i++)
                room[i][15]=0x001F;

/* obstacle 7 */
for (i=209;i<=229;i++)
        for (j=16;j<=20;j++)
                room[i][j]=0xFFFF;

/* obstacle 8 */
for (i=35;i<=229;i++)
                room[i][20]=0x7FFF;
for (i=35;i<=229;i++)
                room[i][21]=0xF800;

/* obstacle 9 */
for (i=275;i<=469;i++)
                room[i][20]=0x7FFF;
for (i=35;i<=229;i++)
                room[i][21]=0xF800;
/*for (i=90;i<115;i++)
        for (j=0;j<21;j++) {
printf("%x ",room[i][j]);
if (j)
if ((j%20)==0)
        printf("\n");
        } */
}
```

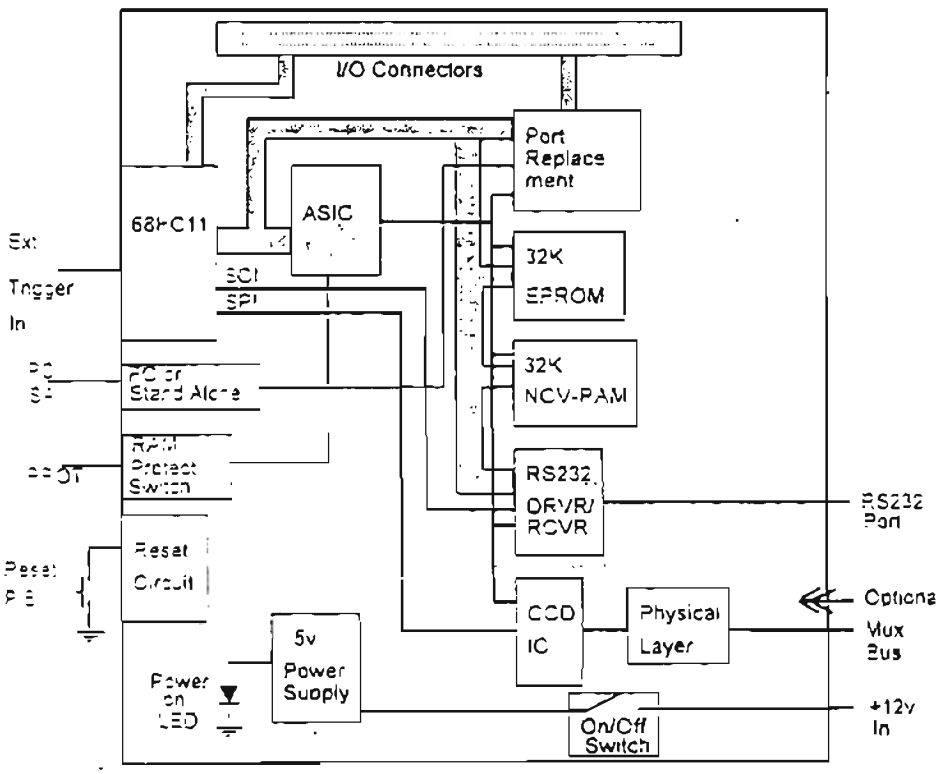
APPENDIX G

DIAGRAMS OF THE ENAT BOARD

COMMENTS FOR THE APPENDIX

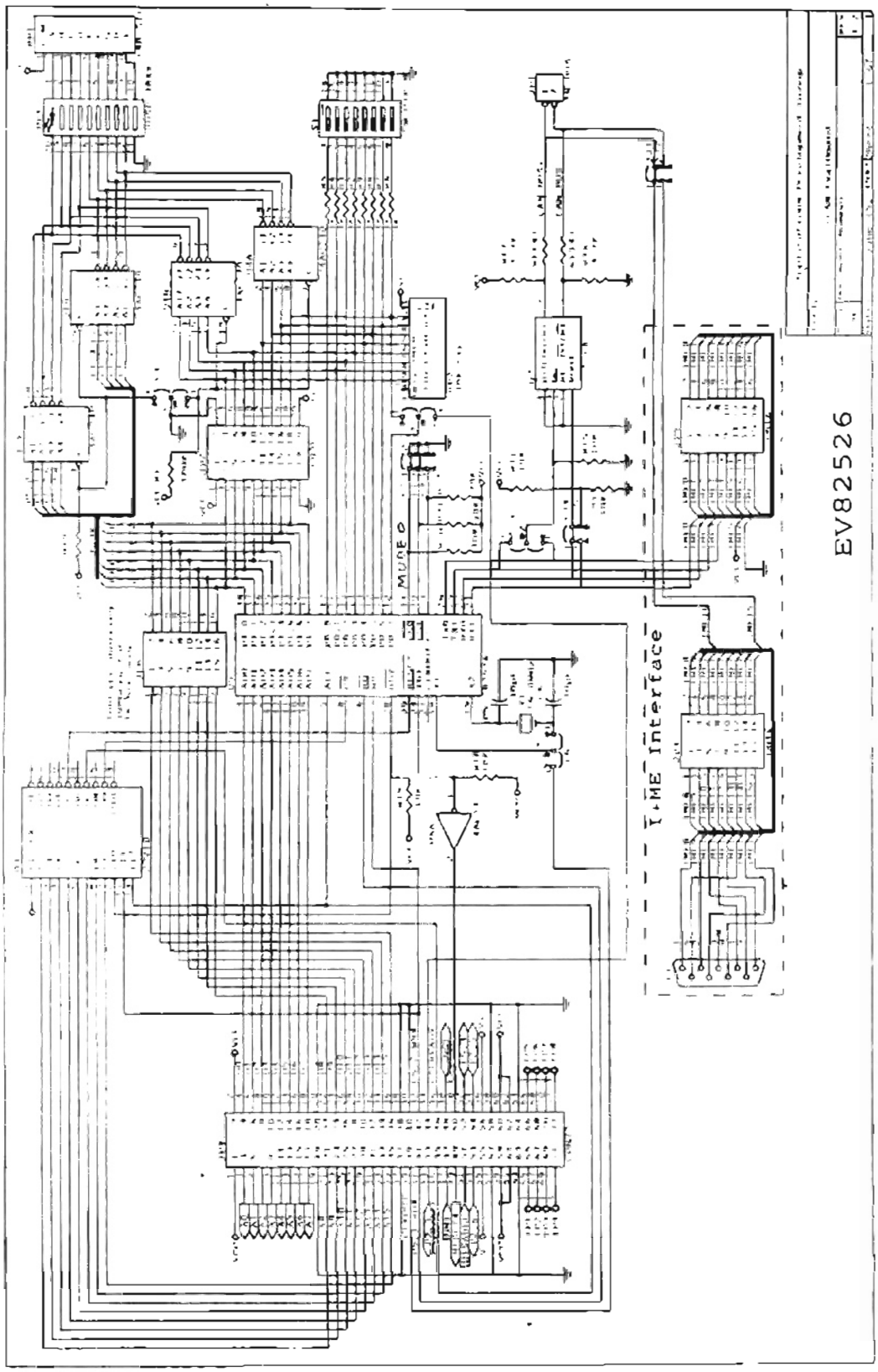
The main purpose of this appendix is to collect and show in one place the important diagrams of the ENAT board that are not shown and do not appear in the documentation (User's Manual) of the board.

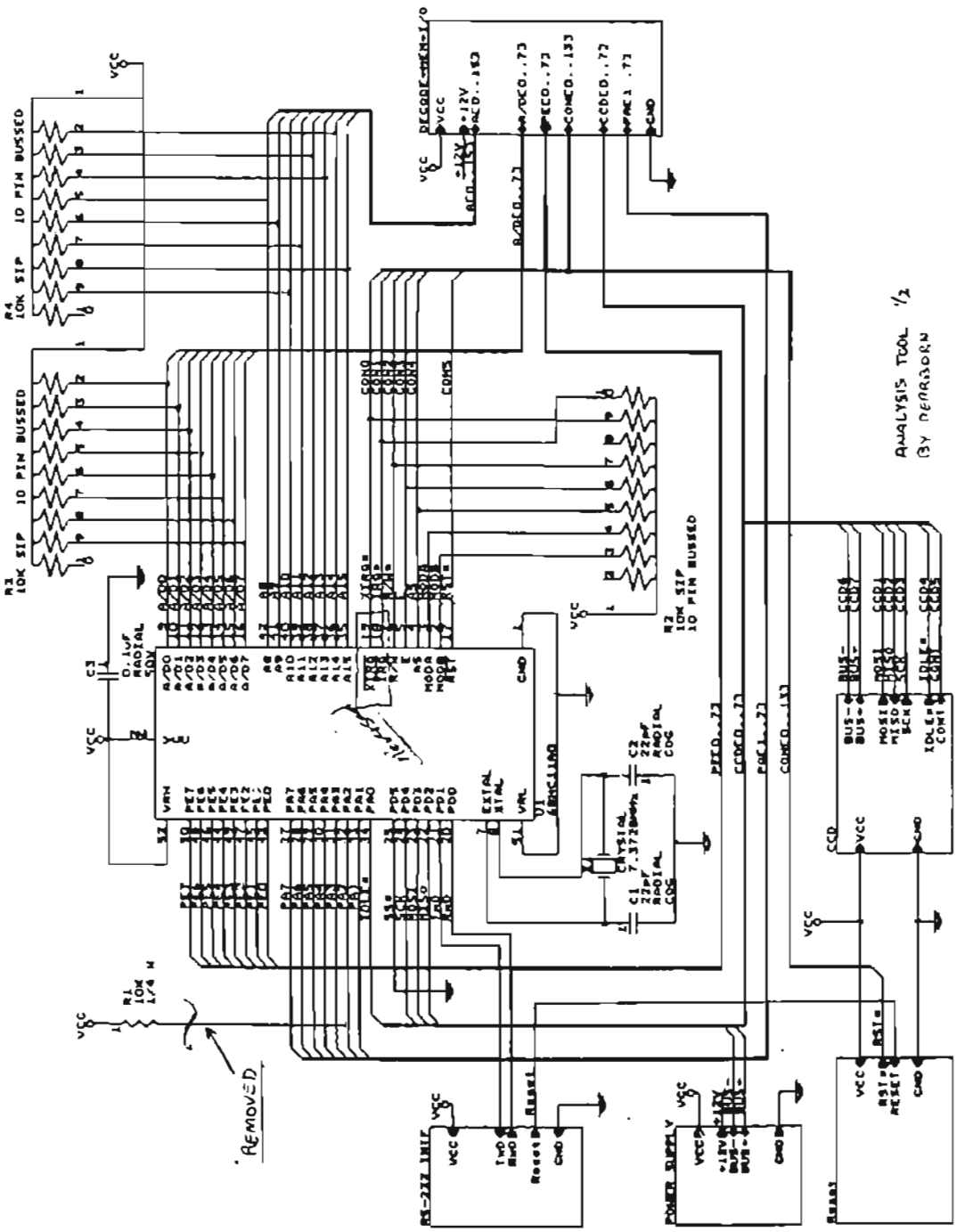
Also other objective of this appendix is to indicate the modifications done to the Propulsion controller. The elements that were disconnected are indicated.

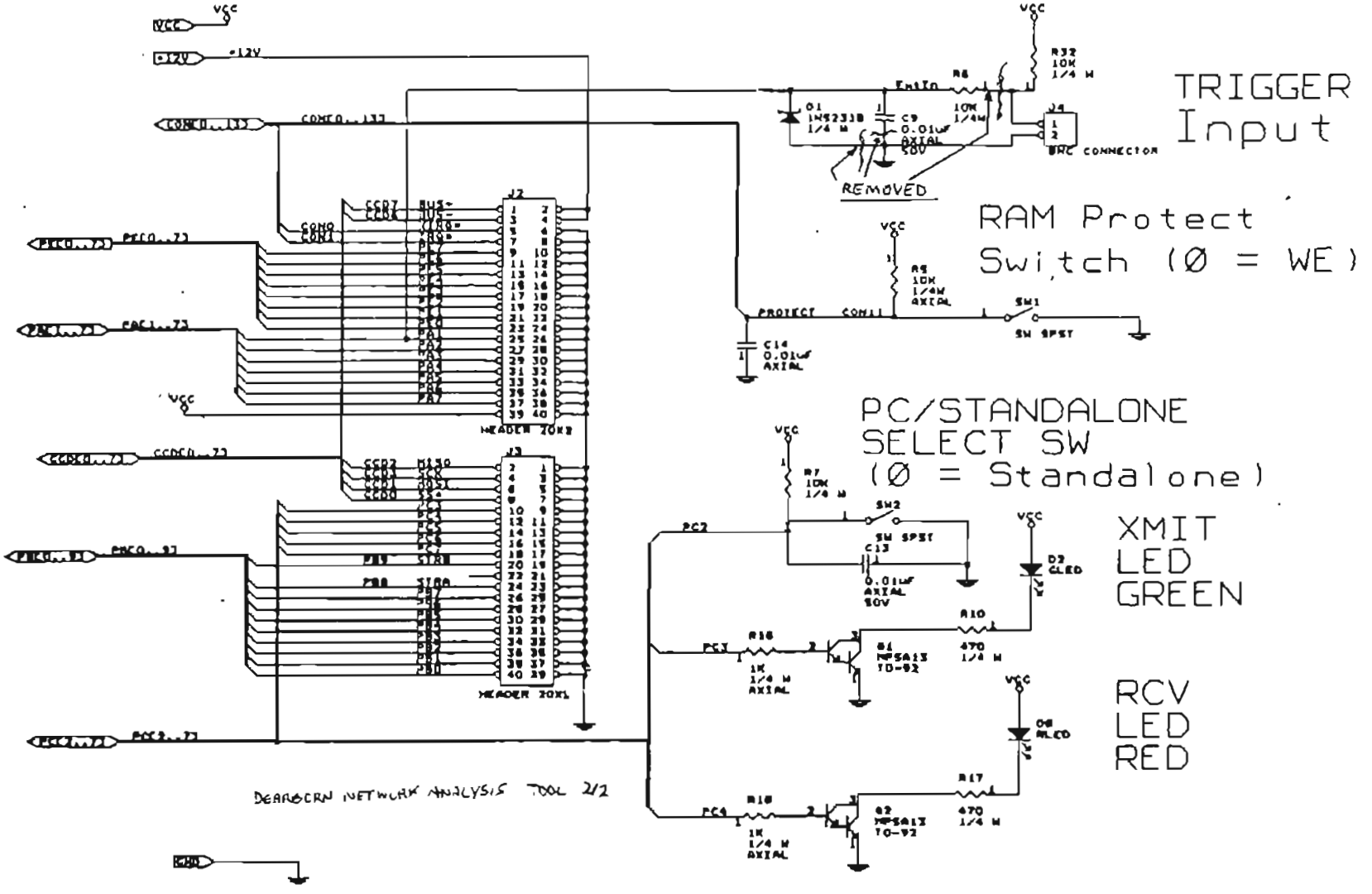


DNAT Block Diagram

82527 Daughter Board




ANALYSIS TOOL 1/2
BY DEFENDORN
REMOVED


TRIGGER
Input
RAM Protect
Switch (Ø = WE)
PC/STANDALONE
SELECT SW
(Ø = Standalone)
XMIT
LED
GREEN
RCV
LED
RED
REMOVED
DEARBORN NETWORK ANALYSIS TOOL 2/2

VITA

Enrique M. Acuña

Candidate for the Degree of

Master of Science

Thesis: DESIGN AND INTEGRATION OF HARDWARE AND SOFTWARE TO IMPLEMENT AN AUTONOMOUS VEHICLE

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in San José, Costa Rica, on March 22, 1968, the son of Carmen Acosta and Enrique Acuña.

Education: Graduated from Calasanz High School, San José, Costa Rica in Nov. 1984 and from Stevens Point Senior High School, Stevens Point, WI, USA in May 1985; studied at National School of Computer Electronics (ENCE) in 1985 through 1986; received Bachelor of Electrical Engineering degree in April 1991 and a "Licenciado" of Electrical Engineering degree in February 1993 both from University of Costa Rica; also has studies of the undergraduate programs of the English Major and the Modern Languages School from the same university. Completed the requirements for the Master of Science degree with a major in Electrical Engineering at Oklahoma State University in May 1996.

Experience: Instructor of the Electricity Workshop at the Center for Community Development, Río Azul, Costa Rica (1987-88); teaching assistant at the Electrical Engineering and Physics Schools of the University of Costa Rica (1988-91); engineering assistant at the Automatic Control Department of SIEMENS S.A. in San José, Costa Rica (1990); Electrical Engineer 2 at Geothermic Plants Office and at Communications Engineering Department of the Costa Rican Institute of Electricity in San José, Costa Rica (1992-93).