

ENVIRONMENT MAPPING AND ADAPTIVE FUZZY
LOGIC CONTROL FOR AN AUTONOMOUS
VEHICLE

By

ROBERT LOREN SHANLEY III

Bachelor of Science

Purdue University

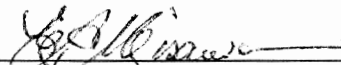
West Lafayette, Indiana

1992

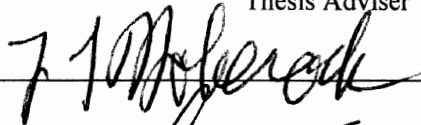
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1994

ENVIRONMENT MAPPING AND ADAPTIVE FUZZY
LOGIC CONTROL FOR AN AUTONOMOUS
VEHICLE


Thesis Approved:



Thesis Adviser







Dean of the Graduate College

ACKNOWLEDGMENTS

My most sincere appreciation goes to Dr. Eduardo Misawa for his advice, good influence, and intensive courses throughout my graduate program. I also thank Dr. Young and Dr. Hoberock for serving on my committee. Their suggestions and support were very helpful throughout this study.

To Robert Taylor and Scott McClain for their encouragement and advice concerning more than just school work.

To Mike Arrington who saved me hours with his advice during those long brain storming sessions. Without his assistance, this project would have taken much longer to complete.

To my parents, Robert Shanley II and Mary Ann Shanley, who kept me going through those difficult years in college. I would have never made it without them. I love you very much.

And to some of the closest people in my life: Stephanie Norton, Jonathan Hynson, and A.J. Schwidder for showing me the way to God.

TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION.....	1
2. LITERATURE REVIEW.....	3
Learning Control Verses Adaptive Control.....	3
Hebbian Learning Rule.....	3
Adaptive Fuzzy Systems.....	4
3. AV CONFIGURATION.....	5
AV Hardware Configuration.....	5
Computer Configuration.....	7
Navigation.....	7
Vision.....	8
Bump Sensors.....	8
Supervisor.....	10
4. BIT FIELD MAP.....	11
Old Screen Pixel Map.....	11
Collection of Sonar Data.....	11
Requirements for the Map Code.....	12
The Bit Field Map.....	13
Map Alteration.....	13
Map Limitations.....	16
Bit Field Map Simulation.....	17
Dynamic Mapping in the AV Simulation.....	18
5. NEW FUZZY LOGIC CONTROLLER.....	19
Brief Introduction to Fuzzy Logic Systems.....	19
Membership Functions.....	19
Rule Base.....	20
Inference Rules.....	26
Max - Min Composition Rules.....	27
Max-Product Composition Rule.....	27
Defuzzification.....	28
The Gaussian Fuzzy Logic Controller.....	28
6. ADAPTIVE FUZZY LOGIC CONTROLLER.....	31
Back-Propagation Training Algorithm.....	31
Implementation of Adaptation Algorithm to AV Simulation.....	34
7. SIMULATION RESULTS AND DISCUSSION.....	36
Non Performance Results.....	36
Performance Results.....	37

Trial #1.....	39
Trial #2.....	42
Trial #3.....	45
Trial #4.....	48
Trial #5.....	51
8. CONCLUSIONS AND FUTURE WORK	54
Future Work	55
References	56
APPENDIX A: MAP SIMULATION SOFTWARE CODE AND DOCUMENTATION.....	57
APPENDIX B: ADAPT SIMULATION SOFTWARE CODE.....	74

LIST OF TABLES

Table	page
1. Comparison of Executable Memory Requirements.....	36
2. Time Required to Compute Fuzzy Logic Output.....	37

LIST OF FIGURES

Figure	Page
1. The Autonomous Vehicle.....	5
2. Instrumentation for speed and position.....	6
3. Location of Blind Zone Around the AV.....	9
4. Schematic of the Bump Sensors	10
5. Configuration of Sonars and their Output.....	12
6. Triangle Configuration within Map.....	15
7. Clearing Horizontal Triangles.....	16
8. Graphic Representation of the Bit Field Map.....	17
9. Real Numbers Close to 10.....	19
10. Location of Waypoint Relative to Robot.....	22
11. Distance of Waypoint Relative to Robot.....	22
12. Desired Speed of Robot.....	23
13. Desired Steering Angle of Robot.....	23
14. Sensor Distance to Obstacle Membership Functions.....	25
15. Speed Change Membership Functions.....	25
16. Steering Change Membership Functions.....	26
17. Way Point Angle Fuzzy Relations.....	30
18. Way Point Distance Fuzzy Relations.....	30
19. Network Representation of the Fuzzy Logic System.....	31
20. Definition of Way Point Angle.....	32
21. Generated Path for Trial One.....	39
22. Way Point Angle for Trial #1	40

23. Steering Angle for Trial #1.....	40
24. Cost Function for Trial #1.....	41
25. Generated Path for Trial Two.....	42
26. Way Point Angle for Trial #2.....	42
27. Steering Angle for Trial #2.....	43
28. Cost Function for Trial #2.....	43
29. Generated Path for Trial Three.....	45
30. Way Point Angle for Trial #3.....	45
31. Steering Angle for Trial #3.....	46
32. Cost Function for Trial #3.....	46
33. Generated Path for Trial Four.....	48
34. Way Point Angle for Trial #4.....	48
35. Steering Angle for Trial #4.....	49
36. Cost Function for Trail #4.....	49
37. Generated Path for Trail Five.....	51
38. Way Point Angle for Trial #5.....	51
39. Steering Angle for Trial #5.....	52
40. Cost Function for Trial #5.....	52

CHAPTER I

INTRODUCTION

Autonomous vehicles (AVs) are currently used in many environments where human intervention is not possible. These applications include toxic waste disposal, the physically disabled, manufacturing plants as well as sea and space exploration. The problem most generally encountered is the need for AV's to know their constantly changing environment. This requires an AV which has the ability to create and change a map of its environment and to alter its controller for the conditions at hand.

Currently, the AV simulation program, [1] simulates two micro controllers and a supervisor. Each micro controller will control a different aspect of the robots movement and data acquisition. For instance, one micro controller will be used to collect data from the vision sensors, and a second micro controller may be used for the actual control of the propulsion system. The supervisor will essentially be the brain of the system by providing commands such as desired speed and heading angles. The supervisor must conform to the J1939 Controller Area Network (CAN) standard [2], for communication between the modules. Additional operations the supervisor performs are a sensor based path planner and mapped path planner developed by [1] to provide the optimal path to the desired target location and fuzzy inference rules for obstacle avoidance.

While the simulation works, it does have two problems: The map and the fuzzy inference rules. The map used by [1] was a pixel based screen map. Depending on the color of the pixel, that location is either free, occupied or has unknown obstacles. The problem is the robot will not have a computer screen to store the map information. The other problem is the robot's fuzzy rules are unadaptive to the environment.

It is the aim of the author to implement a new map and develop two new controllers. The map concept developed by [1] will be implemented in a bit field array. This bit field map will be stored in a multidimensional array of integers and will resemble the pixel field map. Next, the FLC will be changed to a different format to make adaptation easier. This new format will also save computer computation time and memory. I will also introduce an appropriate cost function to be used in a back propagated adaptive

fuzzy controller [3,4]. This will allow the robot to adapt its fuzzy inference rules only in the path planner. I will leave the obstacle avoidance for a separate project.

CHAPTER II

LITERATURE REVIEW

Learning Control Verses Adaptive Control

There are many reasons for why an learning control scheme might be needed in a system. According to [10]:

"A learning control system is one that has the ability to improve its performance in the future, based on experimental information it has gained in the past, through closed loop interactions with the plant and its environment."

In other words, learning control algorithms are needed when the conditions are uncertain or it is impractical to create a controller for your system with a priori knowledge. This system is different from adaptive control schemes in that the control law is retained as a function of the operating conditions. That is, depending on the conditions of your system, a different control scheme is used.

Hebbian Learning Rule

One of the main purposes of any control system is to provide robustness to the system. One of the advantages of using fuzzy logic is the ability to apply it to systems which are extremely complex, have a huge number of state variables, or for a system which is ill-defined or unknown.

When implementing a fuzzy logic controller (FLC), some sort of prior knowledge is needed for the system. This could come from a human expert who is able to control the physical system or from the design engineers. The problem is when the control rules are not well defined. Then there is the case when the system deviates from normal operating conditions. If this deviation was unexpected, there probably won't be any rules to compensate the system.

A solution is to apply some sort of learning rule to the system. The fuzzy Hebbian learning rule has been introduced and applied by [11]. Based upon signal Hebbian learning, the system will predict the state of the system forward in time. This information is then used to adjust the parameters of the FLC.

Adaptive Fuzzy Systems

Adaptive fuzzy controllers are different in that they have an additional two components in the system--a process monitor and an adaptation mechanism. The process monitor itself is either a performance measure or a parameter estimate. That is the monitor will either use some measurement of performance (such as overshoot, rise time, settling time etc.) to alter the control parameters, or the monitor will estimate some unknown states or system parameters.

There are several types of adaptation mechanisms as well. The first two types are categorized into the self tuning category. They include routines which will either use scaling factors to change the membership functions or will actually alter the shapes of the membership functions themselves. The other type of adaptation mechanism is the self-organizing type which will actually change the rules of the fuzzy system.

A major question concerning fuzzy systems in general as well as adaptive fuzzy systems is their stability. This is important because we don't want the adaptation routines to cause a stable system to go unstable. However, stability is still an open question when related to adaptive FLC (fuzzy logic controllers) because little work has been done in this area [12].

Li-Xin Wang [4] has shown that certain types of fuzzy logic systems work as universal approximators. If the user is given an input - output pair (x, d) , it is possible to find a fuzzy function $f(x)$ such that the cost function

$$1. \quad e = \frac{1}{2} (f(x) - d)^2$$

is minimized. This is generally considered as a nonlinear function approximator as it will approximate the nonlinear function which produced the output d .

I will introduce and show that this type of adaptive fuzzy logic controller may be used as a guidance parameter identification system within an adaptive fuzzy logic controller. A particular cost function will be developed along with the adaptive functions for the control parameters.

CHAPTER III

AV CONFIGURATION

AV Hardware Configuration

As was stated in the introduction, there are many uses for AV's. One such use is for the physically disabled. Quadriplegics could use an autonomous wheelchair for getting around during their everyday routines. This wheelchair would be able to accept commands as to where to go, and would proceed to that location avoiding obstacles along the way.

After looking at several configurations available to the research team, it was decided the wheelchair was the best option (see Figure 1). This was for several reasons. The first was that motorized wheelchairs already exist and would require little alteration to be applied to our application. The other main consideration was the demand for such wheelchairs for the physically disabled.

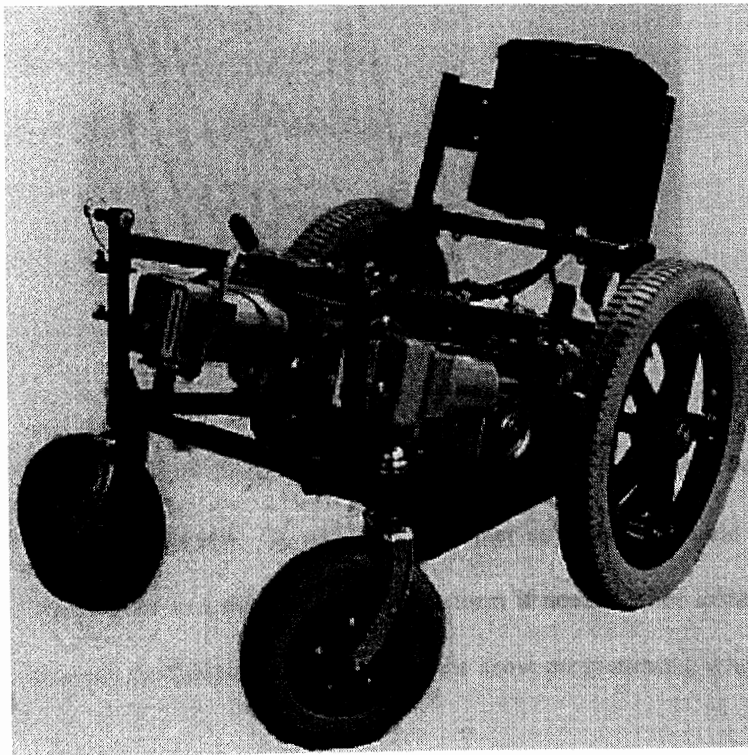


Figure 1: The Autonomous Vehicle

As stated, the wheelchair needed little alteration to be used as an AV. By purchasing a motorized wheelchair, we had the motorized platform already constructed. The batteries and motors were already attached and functioning properly. There is also a working joystick controller available to the user. The seat was removed for convenience, but could be replaced in the future. The only alterations were the addition of speed and position sensors and the computers to control the system (see Figure 2).

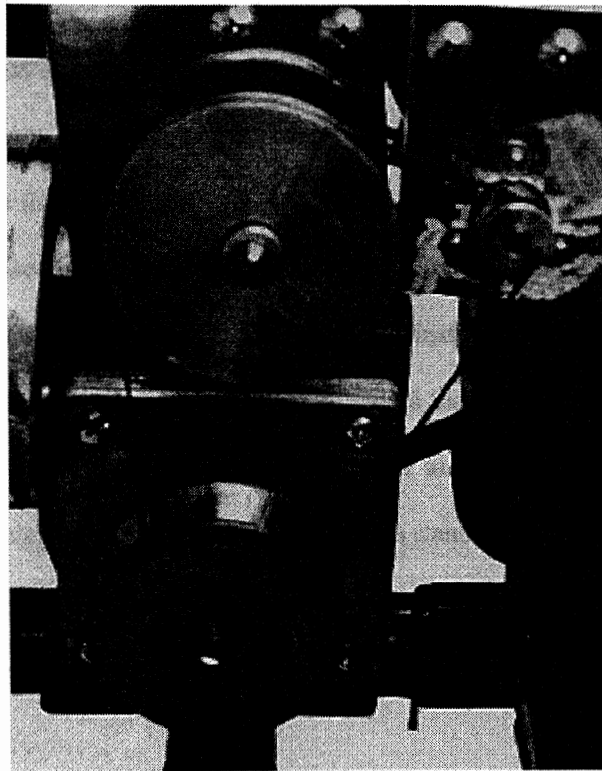


Figure 2: Instrumentation for speed and position

This allowed the research team to save a lot of construction time and will provide an AV for which there is a definite demand in industry. In addition, any other applications would use similar computer equipment and could be adapted to a different mobile platform if needed. For instance, a deep sea probe could use the same computer configuration and some of the same programming while utilizing a different mobile platform.

Computer Configuration

As stated there will be several levels to the computer system of the AV: navigation, vision, bump sensors, and the supervisor. Each module will be connected through a CAN so communication of sensor data and commands from the supervisor are received by the target boards.

Each of the modules, except the supervisor, will be implemented on a Motorola 68HC11 micro controller. The HC11 was chosen because of the data acquisition capabilities built into the chip, removing the need for data acquisition boards. In addition, the HC11 has already been implemented with the J1939 CAN by several other groups in industry and scholastic professions. The supervisor will be installed on a 486DX40 clone. This was for several reasons. The first is the memory limitations on the HC11. The map alone will require more memory than is available on the micro controllers. The PC's today can have megabytes of memory on the motherboards themselves, eliminating any memory restrictions. A second reason for the PC is a hard drive may be added to provide a black box recorder. This would provide a detailed description of the operations of the AV and could also provide storage for multiple maps of different environments. In addition, if the AV is to become a stand alone system, it would need storage for the source code in case a reboot is needed. The HC11's do not provide for disk storage capabilities and thus would not make a good choice for the supervisor.

Navigation

The navigation is used for data acquisition of speed, and position as well as control of the propulsion systems. The data for position and speed are collected through encoders and tachometers respectively. The tachometers and encoders are connected to the shaft of each rear drive wheel of the AV through a belt similar to the timing belts used in today's automobiles. This will provide the distance traveled and speed of each drive wheel which the controller will use to determine corrections to the motor inputs.

Vision

The vision is used by the supervisor for obstacle avoidance and to create the map. The vision module is made of six Polaroid sonars: three out the front, and one out each side and the back. The three sonars on the front will have one pointing straight out the front and two pointing out at some angle to provide a maximum resolution in the direction of travel. The reader is directed to [5,6] for further information concerning the vision module.

The vision module is also used as a damage control device in case the supervisor was to fail during operation. If the vision no longer recognized the supervisor as on-line, it would park the AV in a safe location and wait for further commands.

Bump Sensors

A new module added to the system are bump sensors. A problem was discovered during the first simulation [1] where the sonars would not detect objects within a close proximity (40 cm) of the robot. This is because the sonars ignore all data from 0 to 40 cm because of echo effects inherent in the design. Thus, when an obstacle enters the 40 cm zone of the robot, it does not register the object and would proceed as though it was not there (see Figure 3).

There are several solutions to this problem which were considered. The first was not to use sonars and switch to infra red sensors. This would be very expensive and time consuming. The second solution is to continue to use the sonars and add a bump sensor to the platform of the AV [7]. The bump sensors are very similar to curb sensors on some of the luxury cars today (see Figure 4). When one of the whiskers is deflected by an obstacle, a warning is sent to the supervisor showing the location of the obstacle.

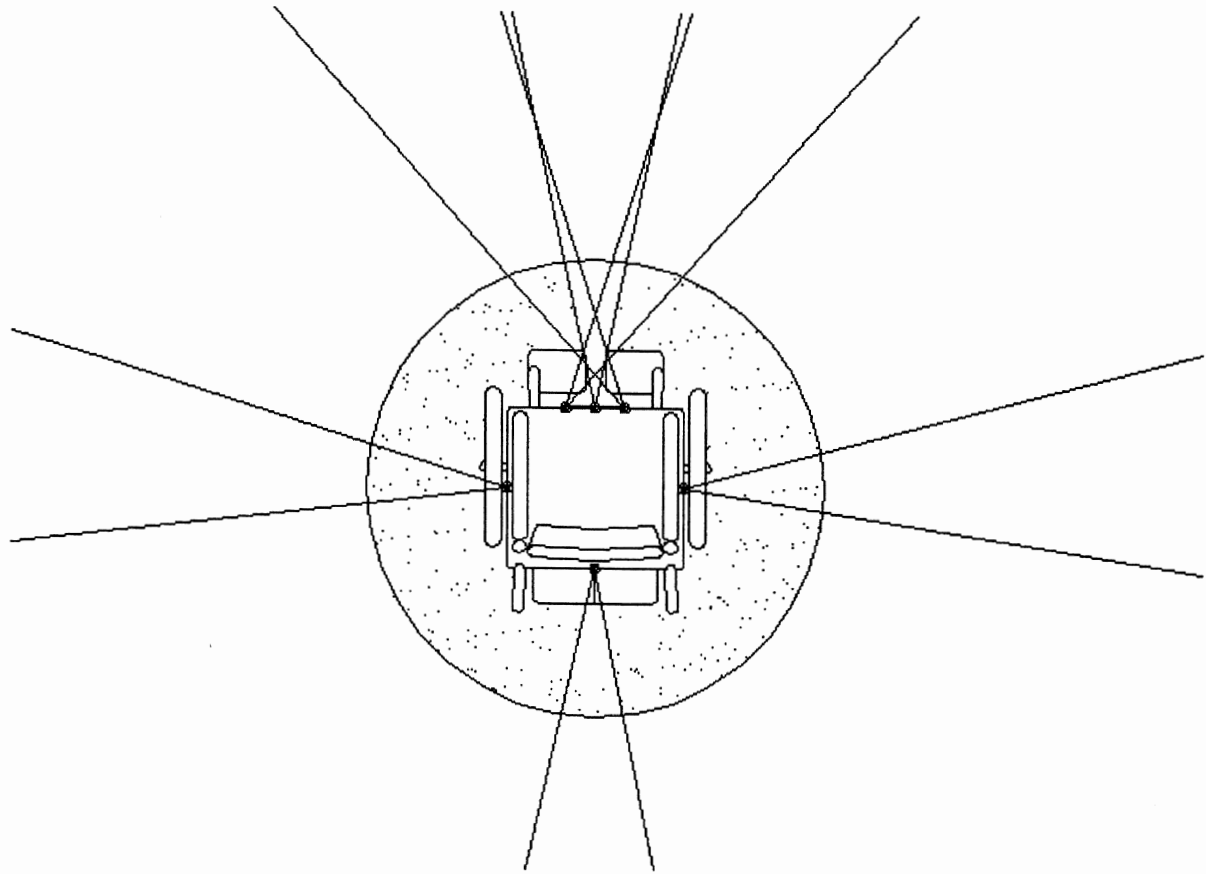


Figure 3: Location of Blind Zone Around the AV
[7]

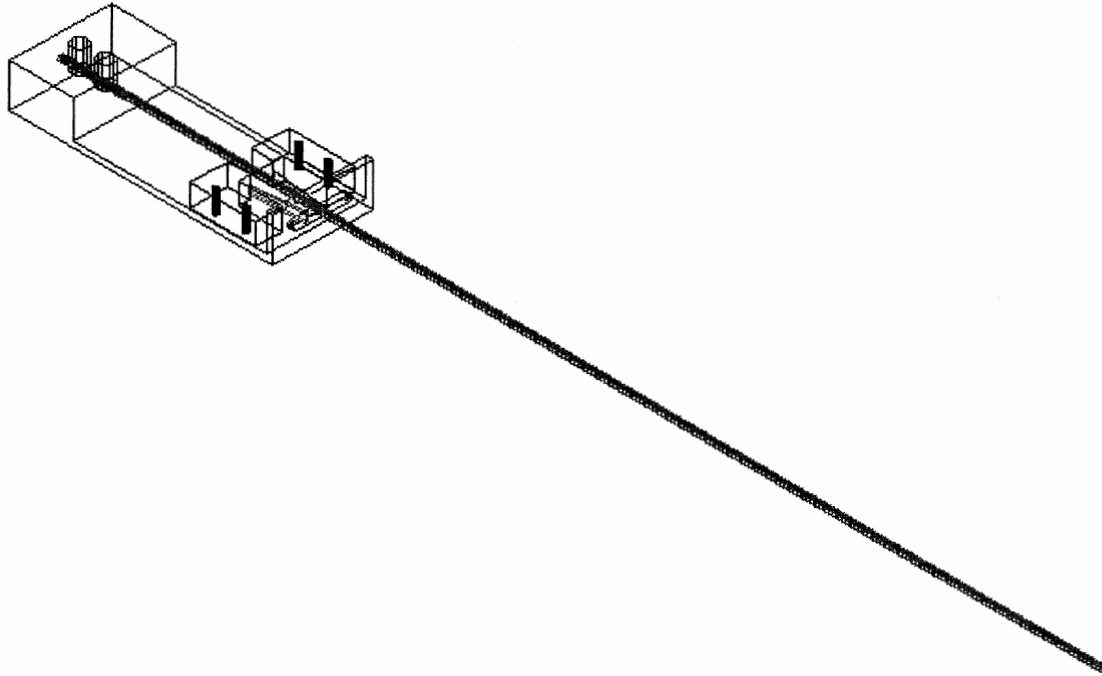


Figure 4: Schematic of the Bump Sensors
[7]

Supervisor

It was said by Dr. Pedro Alberto, the current IFAC vice-president, the supervisory controllers usually deal with complex systems or oversee several systems being controlled by other controllers [9]. In the AV situation we are currently working on, it is what makes the main decisions and creates the environment map. The module uses both memory map and sensor path planning routines to reach the desired location. It also sends the destination data, desired speed and heading angle to the other modules using the CAN. The fuzzy inference rules are used to avoid obstacles and possibly track moving objects when desired. The supervisor also provides a way of communication to the outside world. Currently communication takes place through a keyboard terminal but could be expanded into voice command or some other advanced form of user input.

CHAPTER IV

BIT FIELD MAP

Old Screen Pixel Map

As was mentioned in the introduction, the simulation by [1] used the computer screen to introduce the environment and store the map. The screen allowed three forms of information to be stored in the map: occupied, free, and unknown space. It stored the information in the form of the color of the pixel at that location. For instance, a white pixel was occupied by an obstacle, black pixels were free space, and gray signified an area for which the robot had no information.

As the simulation progressed, the robot would place black patches over the areas the sonars said were unoccupied. While this worked fine for the simulation, it would be unusable during implementation as there are no monitors on the frame of the AV. It was decided by the research team, to alter the simulation's map before we implemented the software into the actual vehicle.

Collection of Sonar Data

Before the map software could be rewritten, a complete understanding is needed of how the sonar information is represented. The sonars return only one piece of information, the distance to the obstacle. This distance does not mean there will always be an obstacle at this location. The sonars have a maximum range available and when there are no obstacles within this range, that maximum distance is returned.

Of course there is other information available to the mapping routines, namely the current location of the AV, heading angle, and the angle between the sonars and the robot frame. With this information, it is possible to describe the area in which the sonar scanned for obstacles.

The next key item about the sonars is the shape of the scanning area. The sonars will scan a room by sending out a pulse of sound. This sound wave will travel out in the shape of a cone with half-arc-angles of 12 or 10 degrees depending on the setting of the instrumentation. It was decided early on to make the new map as similar to the one used previously. This meant we would use a two dimensional map similar to the

screen on the computer. A two dimensional cone becomes a triangle with the same angles as the cone. Now we know everything about the data collected by the sonars.

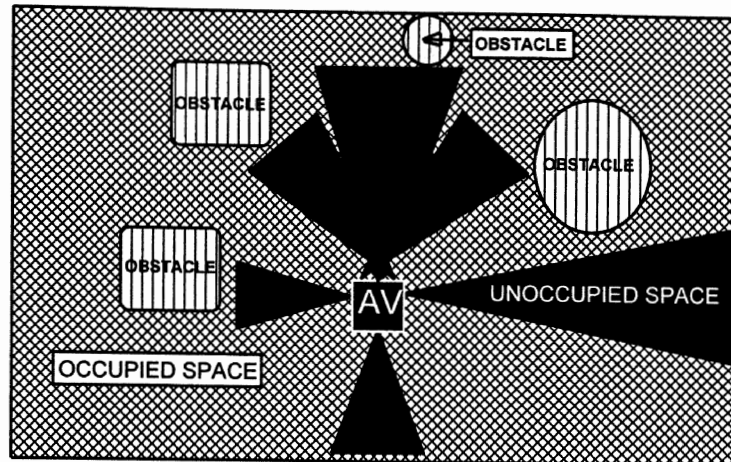


Figure 5: Configuration of Sonars and their Output [1]

Requirements for the Map Code

There were several major concerns about implementing a map into the memory of the robot. The first was a need for the efficient use of memory. Even though it was decided to use a PC board for the supervisor, we don't have unlimited memory available for map storage. There is still a limit of 640 K due to using MS-DOS as the operating system.

The map should be able to work in all directions of the Cartesian plane. The map would not be of much use if it could only make alterations in one direction from a given point. The AV has six sonars mounted on its frame and they all point in different angles depending on the orientation of the robot. Please refer to Figure 3 for an example of the sonar orientation.

The routines which altered the map should also be stand alone (or portable) code. If for some reason the code needed to be changed, it would be nice if the entire map could be removed and replaced with the new algorithm.

The final consideration in choosing a map configuration, is the ability to alter the map at any time. One of the possible uses for this robot is to track and follow moving objects. If you allow moving objects into the robot's environment, it would have to be able to replace "free" space with "occupied" space. Of course, when the robot returns the maximum range as the distance to an obstacle, it shouldn't place an obstacle at this location. As stated, the robot will return the maximum distance when there is no obstacle within that sonar.

The Bit Field Map

It was decided that the best solution to the memory map would be to create memory similar to the screen. Instead of pixels, bits will be used to store the information. The difference is that bits are either on or off so only two conditions can be stored as compared to the three situations stored in the pixel map. The dropped information was the unknown or gray condition. By assuming all space is occupied until the sonar tells the AV different, the map could be represented by a set of '1' or '0' where the '1' indicates occupied space and '0' free space.

Map Alteration

Before discussing the map alteration, the reader must know how the map is defined. A structure called bit was created in the C code which held the column and row bit. Column bits (which represented the x location in the Cartesian plane) were numbered consecutively from 0 to the maximum number of bits. The row bits (representing the y location) was defined in a similar manner. This would mean that the top left bit of the map is defined as bit (0,0). It is very similar to how Cartesian coordinates are represented as (x, y) except that the bit is defined as (column, row).

Each bit is also, as stated, part of an integer. Thus in order to change a given bit, the program must first find the appropriate bit number using the equation,

$$2. \quad a = \text{zero.col_bit} + \frac{x}{\text{resolution}}$$

where zero.col_bit is a structure containing the starting location of the robot in bits. That is it is analogous to the location of the robot in a Cartesian plane using the units of bits. X is the current location of the robot and resolution is defined as the number of meters per bit. In other words, equation (2) finds the distance between the current location and the reference location and adds this distance to the reference location giving the current location in bits. Note that the ratio in (2) has to be converted to an integer with the proper rounding. The y bit is found in a similar manner except we now use zero.row_bit and y. Next we must be able to define the integers within the map array which contains that bit with the equation

$$3 \quad \text{int} = (\text{int})\text{floor}\left(\frac{\text{col_bit}}{\text{s_byte}}\right).$$

for the x location where col_bit is the bit referencing the x location and s_byte is an integer referring to the size of a byte (16 on a 32 bit machine). The floor function in BorlandC [8] returns the largest integer which is less than the argument. The y location is much simpler as it is simply equal to the row_bit.

Once the integer containing the desired bit has been defined, we can isolate the desired bit by creating the appropriate mask. In most cases, there is more than one bit within an integer which needs to be cleared. Let us consider the case when a few bits within an integer need to be cleared but there are also several bits within that integer which need to remain occupied. The mask for this situation is found with the following loop:

```
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));
```

where l_bit refers to the left most bit and r_bit is the right most bit within the integer a (note that a equals c), and j is the mask being created and its initial has all bits equal to 1. For the case when the row to be cleared spans multiple integers, we first clear the left most integer with the mask:

```

/* 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));
and then clear the integer containing the right most bit with the mask:

```

```

/* clear right bit */
for (i=0;i<=r_bit.col_bit-(c*s_byte);++i)
    j=j-(int)pow(2,(s_byte-1-i));

```

then the whole integers are cleared in between the right and left if they exist.

The procedure for altering the map is fairly straight forward. The first step requires the definition of the triangle produced by the sonars. Using basic geometry and knowing the height and angles of the triangle, as well as the base point (the origin of the sonar), the other two points could be found thus defining the lines between the three points. Please refer to Figure 6 for triangle definition.

The robot would then take each row of bits inside the triangle, find the right and left most bits for that row, and clear (or make '0') that row of bits. The program would proceed in this manner until the top (or bottom) of the triangle is reached, thus clearing out the entire cone of occupied space.

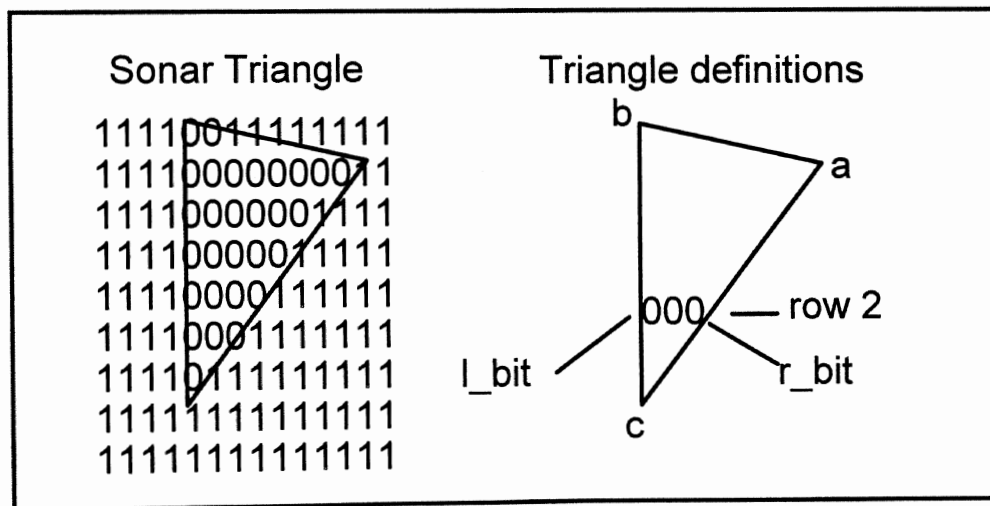


Figure 6: Triangle Configuration within map

The one additional step involves putting '1' on the top line of the triangle if the distance was less than the maximum range, thus signifying an obstacle at this location. Flags within the program would indicate on which side the object is located and whether to place the '1' or not. In all cases except vertical triangles (when the compass heading is either 0 or 180 deg) the '1' would be placed one row at a time as that row was

cleared. In the case of vertical triangles, a whole row would be made occupied after the triangle was completely cleared. This was done because in the case of non vertical triangles, only the end bits would change in the row, were in vertical triangles, the whole row would change.

The horizontal triangles proved to be somewhat of a problem. The horizontal axis of the reference frame cut each horizontal triangle into two separate half triangles. In the case when the triangle was not cut by this plane, the routine could increment one row at a time from the sonar location to the obstacle clearing each row as it went. If this procedure were to be used for the triangles cut by the horizontal plane, only half the triangle would be cleared. The solution dictated that two loops be used where the top half of the triangle would be cleared first, and then the bottom half, thus clearing the entire triangle. Please see Figure 7 for example of clearing horizontal triangles. All triangles can be described and cleared in the two manner of the two described. The only difference is in where the points are located.

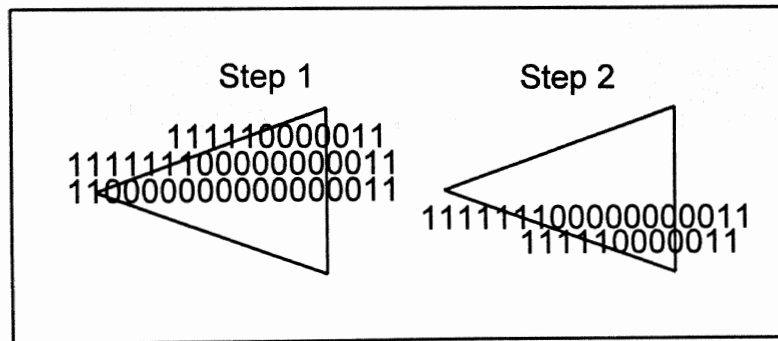


Figure 7: Clearing Horizontal Triangles

Map Limitations

The main limitation of the bit field map is the resolution used to represent each bit. Each bit represents a square area of size dictated by the size of the map to be created. Since BorlandC [8] was used to compile the software, each array had a maximum size of 64K bytes of memory. This means that the map can not take more than this amount of computer memory at any time. Each user of the AV will have to indicate how much area needs to be mapped and decide on the resolution needed in the map.

Bit Field Map Simulation

I have included a photo of the map after alteration, Figure 8. It shows the left have of the simulated environment after the robot has mapped the area.



Figure 8: Graphic Representation of the Bit Field Map

In appendix A, the reader will find the computer code for the map simulation and alteration. This simulation will print a small map on the screen and then prompt the user for location, heading angle, and object distance. The computer will then change the map and display it on the screen in a graphical manner. The reader will also find documentation concerning how the program works. This includes brief discussions on each function used and some of the variables.

Dynamic Mapping in the AV Simulation

Placing the bit field map into the robot simulation written by R. Andujar [1], involved two major steps. The first was just to get the map working. The second step, after completion of the first, was to switch the path planning from the pixel screen map over to the map being created in memory.

In the beginning, we only wanted to get the simulation to generate the memory map and allow the user to view the map. This means that nothing was removed from the original simulation. This allowed me to make any adjustments to the software while the simulation was running. The majority of the adjustments were conflicts in variable types and file names.

Once the simulation was generating the map successfully, I removed all of the screen mapping commands and switched the path planner to the memory map. The screen map itself was left intact because of the need for some kind of input to the system. As a simulation, we still needed to show the environment to the sonars. However, the path planner no longer has access to the screen.

CHAPTER V

NEW FUZZY LOGIC CONTROLLER

Brief Introduction to Fuzzy Logic Systems

Before we can develop an adaptive fuzzy logic controller (AFLC), we have to specify the type of membership functions, rule base, type of inference rules, and the method of defuzzification. These are the basis of any fuzzy logic controller. I would like to refer the reader to the books by Zimmerman [13] and Driankov [12] if he/she is unfamiliar with fuzzy systems. They are good introductory books which cover multiple issues and applications concerning the development of such systems.

Membership Functions

The membership function defines the degree of compatibility or degree of truth given to a specified universe. For example, the membership function shown in Figure 9 could be used to represent all real numbers close to 10. As can be seen, all numbers less than 5 and greater than 15 have zero membership which means they are not close to 10. On the other hand, numbers between 9.75 and 10.25 are considered to be close to 10 and thus have a degree of truth of 1.

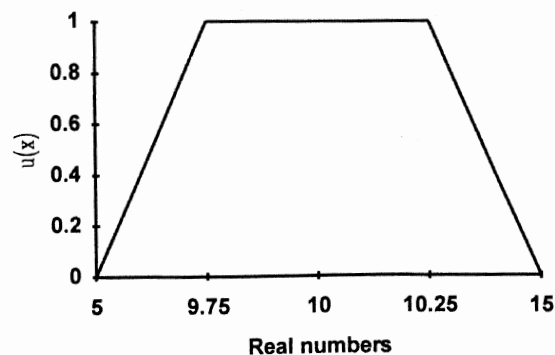


Figure 9: Real Numbers Close to 10

The membership function shown in Figure 9 is of the normalized trapezoidal type. Two other popular options are the triangular and Gaussian membership functions. While there are many reasons for using a particular type of function, they are usually chosen by the user depending on his own liking. The advantage

of using one of the above mentioned functions is the ease for which they can be represented in a computer. For instance, the trapezoidal function can be represented with only four numbers. In the case of the function shown in Figure 9, we could represent real numbers close to 10 with the vector [5, 9.75, 10.25, 15] where each number represents the location on the X axis for that corner of the function.

The triangular and Gaussian functions only need two numbers for representation in a computer. In the triangular case, the user only needs to store the peak and the length of the base. In the case of Gaussian, we only need to store the peak and the standard deviation. This is assuming that both functions are normalized to 1. If the user wished to use unnormalized functions, he only needs to add one number to each. This shows a clear advantage to using triangular or Gaussian over trapezoidal when the amount of memory available is of concern.

Rule Base

The rule base is a set of rules written by the user to take a given input and determine the appropriate output. Rules are generally written by a person who has some experience in the operation of the system trying to be controlled. The rules, when written down, represent a set of if then rules of the form:

If A is close to 10, then output is...

I am using the rules written by [1] from the original simulation. The original simulation utilized two separate controllers. One to keep the AV on the desired path (controller #1), and another to avoid obstacles (Controller #2). The first controller required 12 rules to implement. The rules used are,

```
R1: if (Waypoint Angle == FRONT_OF_CAR AND
      Waypoint Distance == SMALL_DISTANCE)
    then
      Desired Speed = ZERO;
      Desired Steering Angle = STRAIGHT;

R2: if (Waypoint Angle == FRONT_OF_CAR AND
      Waypoint Distance == MEDIUM_DISTANCE)
    then
      Desired Speed = FORWARD_SLOW;
      Desired Steering Angle = STRAIGHT;
```

R3: if (Waypoint Angle == FRONT_OF_CAR AND
Waypoint Distance == LONG_DISTANCE)
then
Desired Speed = FORWARD_MEDIUM;
Desired Steering Angle = STRAIGHT;

R4: if (Waypoint Angle == BEHIND_CAR AND
Waypoint Distance == SMALL_DISTANCE)
then
Desired Speed = ZERO;
Desired Steering Angle = STRAIGHT;

R5: if (Waypoint Angle == BEHIND_CAR AND
Waypoint Distance == MEDIUM_DISTANCE)
then
Desired Speed = REVERSE_SLOW;
Desired Steering Angle = STRAIGHT;

R6: if (Waypoint Angle == BEHIND_CAR AND
Waypoint Distance == LONG_DISTANCE)
then
Desired Speed = REVERSE_MEDIUM;
Desired Steering Angle = STRAIGHT

R7: if (Waypoint Angle == RIGHT_OF_CAR AND
Waypoint Distance == SMALL_DISTANCE)
then
Desired Speed = ZERO;
Desired Steering Angle = HARD_RIGHT;

R8: if (Waypoint Angle == RIGHT_OF_CAR AND
Waypoint Distance == MEDIUM_DISTANCE)
then
Desired Speed = FORWARD_SLOW;
Desired Steering Angle = HARD_RIGHT;

R9: if (Waypoint Angle == RIGHT_OF_CAR AND
Waypoint Distance == LONG_DISTANCE)
then
Desired Speed = FORWARD_MEDIUM;
Desired Steering Angle = HARD_RIGHT;

R10: if (Waypoint Angle == LEFT_OF_CAR AND
Waypoint Distance == SMALL_DISTANCE)
then
Desired Speed = ZERO;
Desired Steering Angle = HARD_LEFT;

R11: if (Waypoint Angle == LEFT_OF_CAR AND
Waypoint Distance == MEDIUM_DISTANCE)
then
Desired Speed = FORWARD_SLOW;
Desired Steering Angle = HARD_LEFT;

```

R12: if (Waypoint Angle == LEFT_OF_CAR AND
        Waypoint Distance == LONG_DISTANCE)
then
    Desired Speed = FORWARD_MEDIUM;
    Desired Steering Angle = HARD_LEFT;

```

Figures 10 to figure 13 show membership functions used by [1] for all the linguistic variables in the first controller. Note that [1] used both triangular and trapezoidal membership functions.

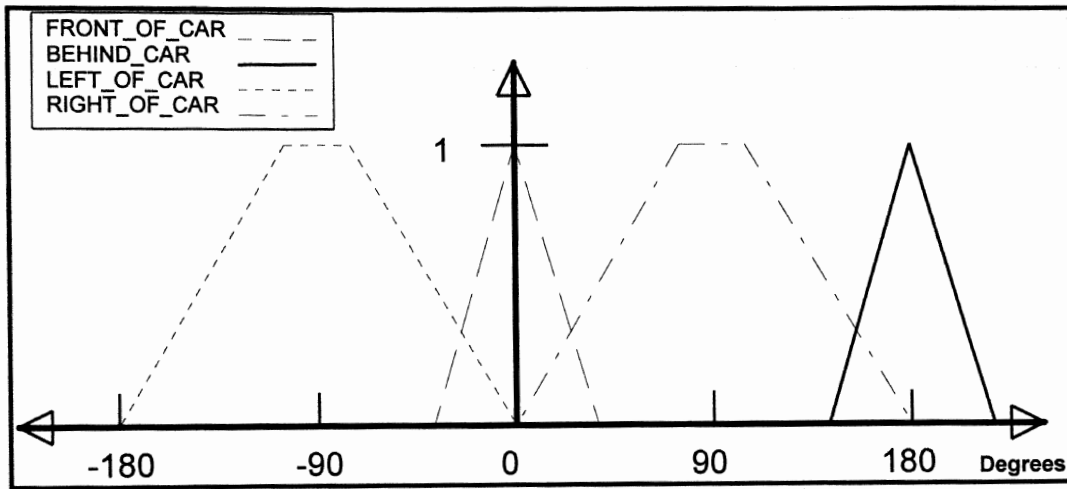


Figure 10: Location of Waypoint Relative to Robot
[1]

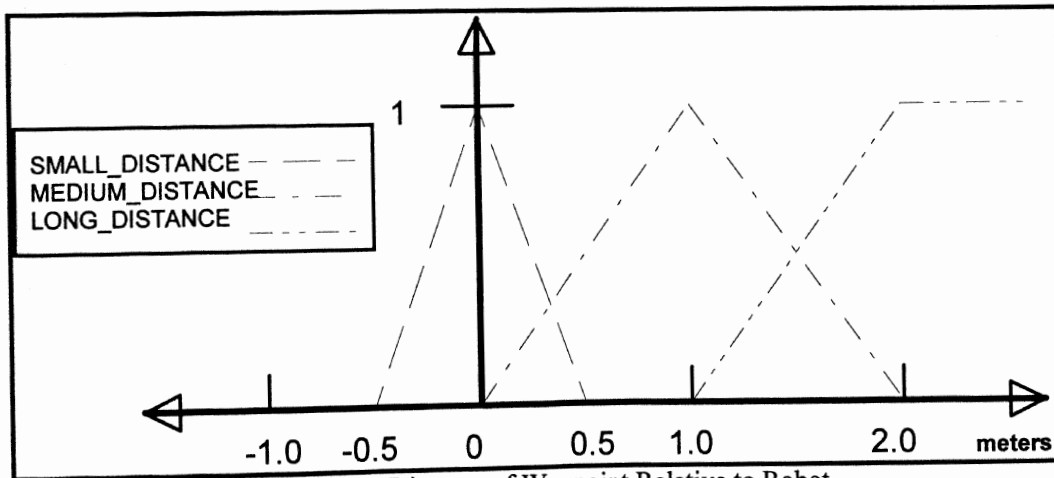


Figure 11: Distance of Waypoint Relative to Robot
[1]

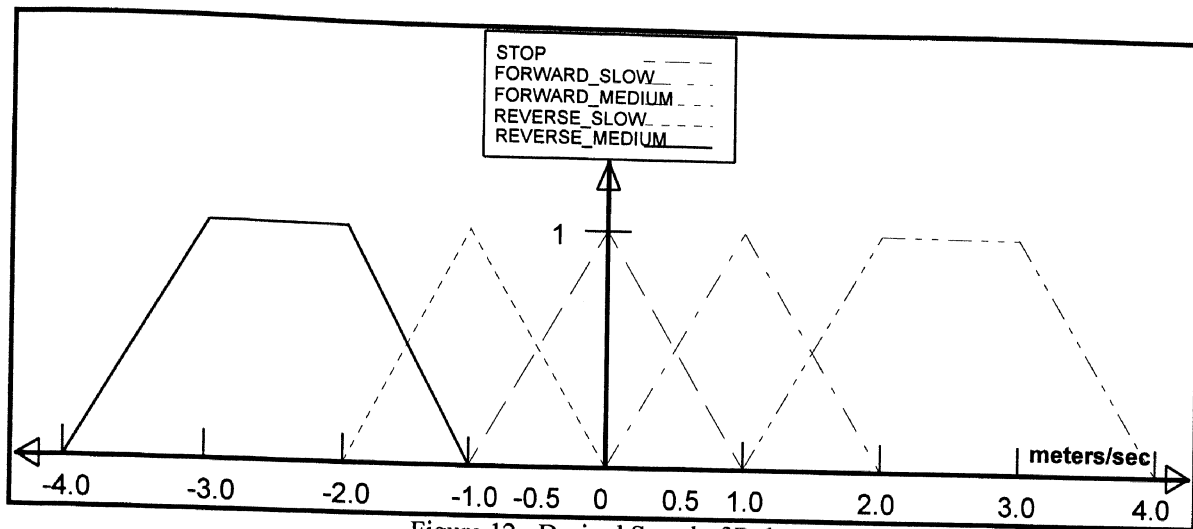


Figure 12: Desired Speed of Robot
[1]

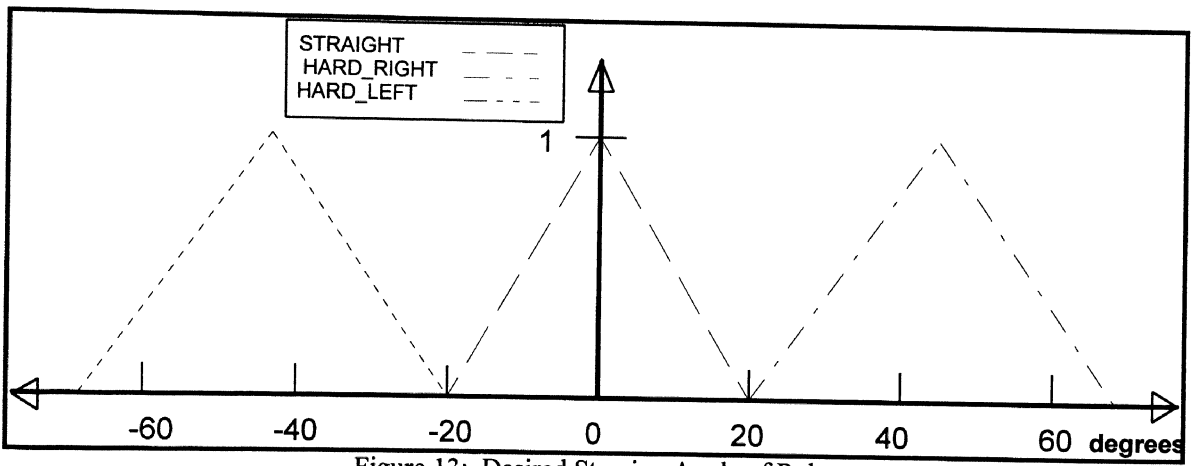


Figure 13: Desired Steering Angle of Robot
[1]

The second controller contains 12 rules. The rules are,

- R1: if(Steering == STRAIGHT AND..Right==VERY_CLOSE)
 - then
 - Steering Change = TO_RIGHT;
 - Speed Change = SLOWDOWN;

- R2: if(Steering == STRAIGHT AND Left==VERY_CLOSE)
 - then
 - Steering Change = TO_LEFT;
 - Speed Change = SLOWDOWN;

```

R3: if(Steering == STRAIGHT AND Front==TOO_CLOSE)
    then
        Steering Change = ZERO;
        Speed Change = SLOWDOWN;

R4: if(Steering == HARD_LEFT AND Right==VERY_CLOSE)
    then
        Steering Change = TO_RIGHT;
        Speed Change = SLOWDOWN;

R5: if(Steering == HARD_LEFT AND Right2==VERY_CLOSE)
    then
        Steering Change = TO_RIGHT;
        Speed Change = SLOWDOWN;

R6: if(Steering == HARD_LEFT AND Front==TOO_CLOSE)
    then
        Steering Change = TO_RIGHT;
        Speed Change = SLOWDOWN;

R7: if(Steering == HARD_RIGHT AND Left==VERY_CLOSE)
    then
        Steering Change = TO_LEFT;
        Speed Change = SLOWDOWN;

R8: if(Steering == HARD_RIGHT AND Left2==TOO_CLOSE)
    then
        Steering Change = TO_LEFT;
        Speed Change = SLOWDOWN;

R9: if(Steering == HARD_RIGHT AND Front==TOO_CLOSE)
    then
        Steering Change = TO_LEFT;
        Speed Change = SLOWDOWN;

R10: if(Steering == STRAIGHT AND Right2==TOO_CLOSE)
    then
        Steering Change = TO_LEFT;
        Speed Change = SLOWDOWN;

R11: if(Steering == STRAIGHT AND Left2==TOO_CLOSE)
    then
        Steering Change = TO_RIGHT;
        Speed Change = SLOWDOWN;

R12: if(Steering == STRAIGHT AND Back==TOO_CLOSE)
    then
        Steering Change = ZERO;
        Speed Change = SLOWDOWN;

```

Figures 14 through figure 16 on the next page show membership functions used by [1] for the second controller.

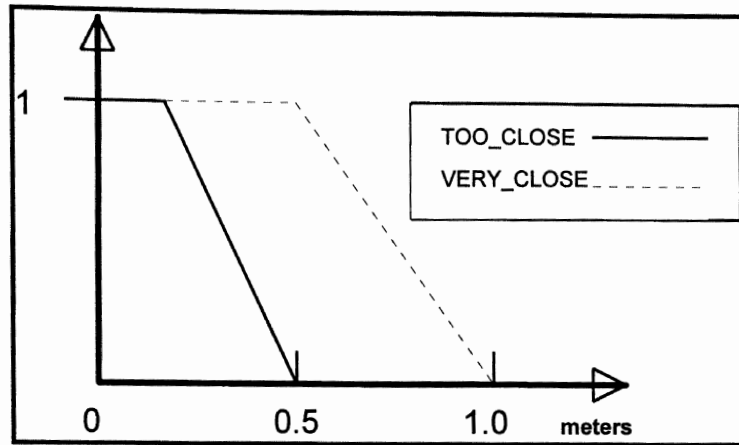


Figure 14: Sensor Distance to Obstacle Membership Functions [1]

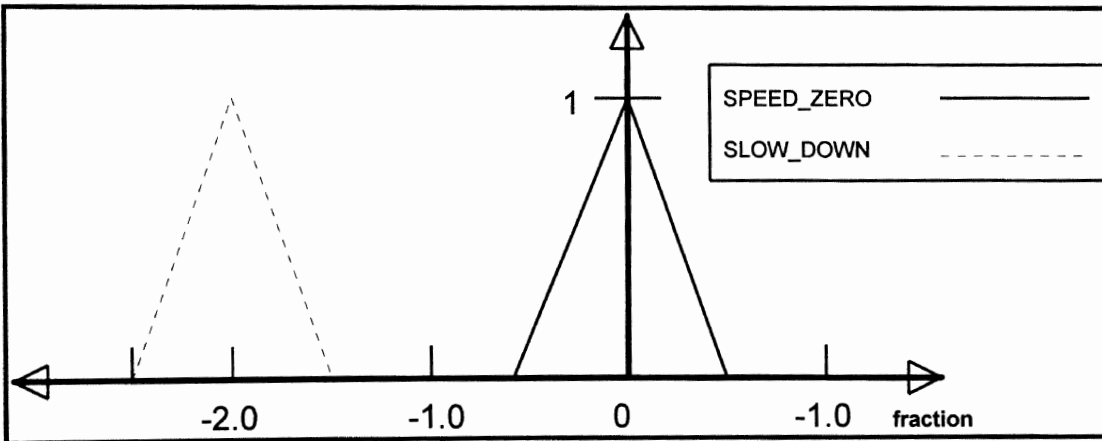


Figure 15: Speed Change Membership Functions [1]

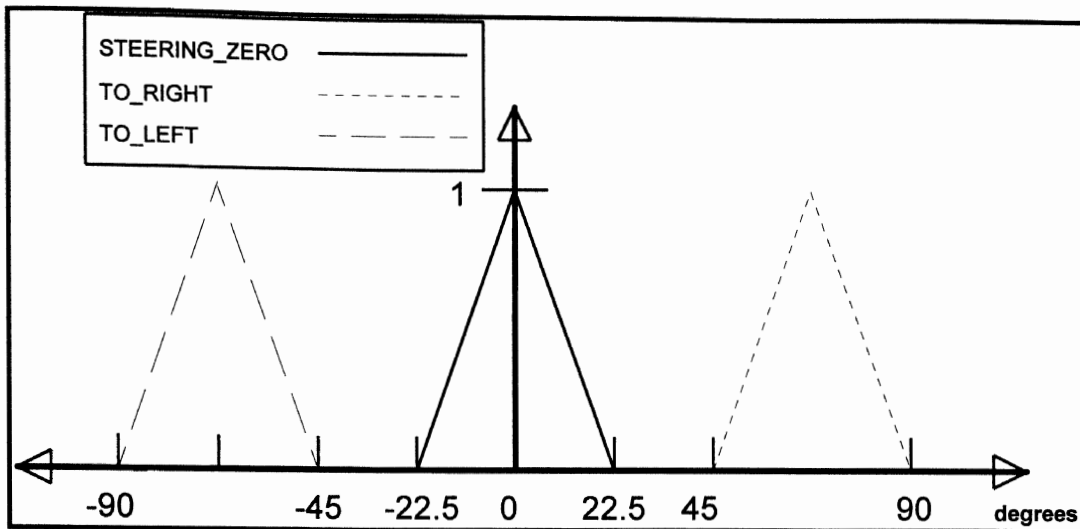


Figure 16: Steering Change Membership Functions
[1]

The methods used to determine how each rule fires and thus which input to use will be determined in the following sections.

Inference Rules

The inference rules take the output of each rule from the rule base and combine them into a fuzzy output for the system. The inference rules used, again is mainly determined by which one the user likes best.

Let us assume, for now, that we are using normalized, Gaussian membership functions and singleton inputs. A singleton input is the sensor reading with a membership of 1. Then let us define a fuzzy number A which has its peak value at a and a standard deviation of b . With these values it is possible to determine the membership or truth factor for a given input from the sensors using equation 4 where x is the measurement from the sensors.

4.
$$\mu = \exp\left(-\left(\frac{x-a}{b}\right)^2\right)$$

By looking at the rules in the previous section, we can see that each rule has at least two inputs (three in the case of Controller #2). I will now look at two inference rules, Max-Min and Larsen's. Instead of going into any theory, I will explain the procedure of using each rule.

Max - Min Composition Rules

Both rules (max-min and Larsen's) will first generate the membership for each input separately. In the case of Gaussian rules, each input will be placed into equation 4 to obtain a membership μ_j , where j is the j th input. Max - Min will then determine the final membership for a given rule i using equation 5 (where equation 4 is already inserted).

$$5. \quad \mu_i = \max_{j=1,m} \left(\exp\left(-\left(\frac{x_j - a_j}{b_j}\right)^2\right) \right)$$

Max-Product Composition Rule

The other composition rule I wish to discuss, is max-product. As stated, you start out as if you are using max-min by determining the individual rule outputs. To obtain the final output membership, we take the product of all the inputs as shown in equation 6.

$$6. \quad \mu_i = \prod_{j=1}^m \exp\left(-\left(\frac{x_j - a_j}{b_j}\right)^2\right)$$

The Max-Min rule was used by [1]. I will use the max-prod rule for reasons to be explained in the following sections.

Defuzzification

The reason for defuzzifying a number is so the computer can send a command to the actuators of the system. Most actuators will only accept crisp or non fuzzy numbers such as a voltage or current. Thus, we want to take the fuzzy outputs generated by the rules and defuzzify them into a single output. Again there are many ways of defuzzifying a fuzzy number. I would like to reference the reader to [12,13] for an in-depth discussion on the different types. I will, however, discuss the center of height and center of area methods which were used in the two different simulations.

The center of area method (used by [1]) will numerically solve for the center of area of the final membership function generated by either equation (5) or (6). The location on the x-axis of C.A. is the final answer sent to the actuators of the system.

The center of height method determines the center of each peak of the final membership function and averages their respective x-axis locations. This method is much faster computationally and generally gives results similar to the C.O.A. method.

The Gaussian Fuzzy Logic Controller

The two controllers I developed for the robot both use the same rules developed by [1]. The difference is that my controllers used Gaussian membership functions, Max-Product with Larsen's rule, and the height method for defuzzification. The second controller, to be discussed in the next chapter, had in addition to the above, adaptation routines to adjust the guidance control parameters.

The membership functions were switch to Gaussian because the exponential functions provide for nice properties and makes for easy integration into an adaptation routine. Combining equation (6) with the center of height method for defuzzification, we can obtain an equation for the crisp output as:

$$7. \quad f(x) = \frac{\sum_{l=1}^m u_l \left[\prod_{i=1}^n \exp\left(-\left(\frac{x_i - a_i}{b_i}\right)^2\right) \right]}{\sum_{l=1}^m \left[\prod_{i=1}^n \exp\left(-\left(\frac{x_i - a_i}{b_i}\right)^2\right) \right]}$$

where u_l is the command output. It is assumed that this is a crisp number as it can be shown that the same results are obtained if a Gaussian membership function were used for the output. This saves some computer memory and computation time over the rules generated by [1]. The former rules had fuzzy output and thus required more computation.

An additional reason for using a fuzzy system of the type shown in equation (7) is because of the Universal Approximation Theorem developed by Li-Xin Wang [4]. The theorem states that fuzzy systems of this type are capable of approximating any nonlinear function to an arbitrary degree of accuracy. The reader is referred to [4] for the proof. This will be discussed further in the next chapter.

Using the Gaussian membership functions, I have tried to obtain about the same amount of area as the original trapezoidal functions. For example, in Figures 17 and 18 I have shown the membership functions for way point angle and way point distance which are the functions used by the path following controller.

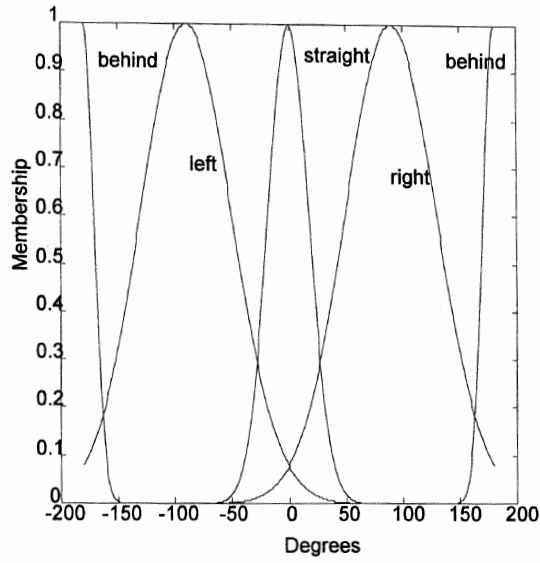


Figure 17: Way Point Angle Fuzzy Relations

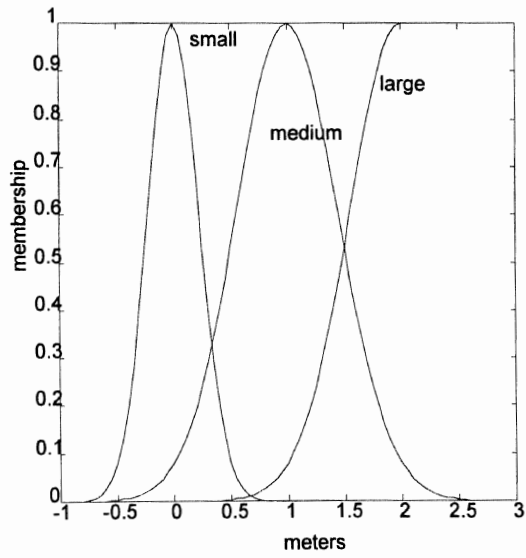


Figure 18: Way Point Distance Fuzzy Relations

CHAPTER VI

ADAPTIVE FUZZY LOGIC CONTROLLER

Back-Propagation Training Algorithm

In Li-Xin Wang's work [3,4], he showed that by observing the functional form of the fuzzy system shown in equation (7), it can be represented as a three-layer feed forward network shown in Figure 19. This representation of the fuzzy logic system creates a straight forward approach to apply the back-propagation algorithm to adjust the parameters in equation (7).

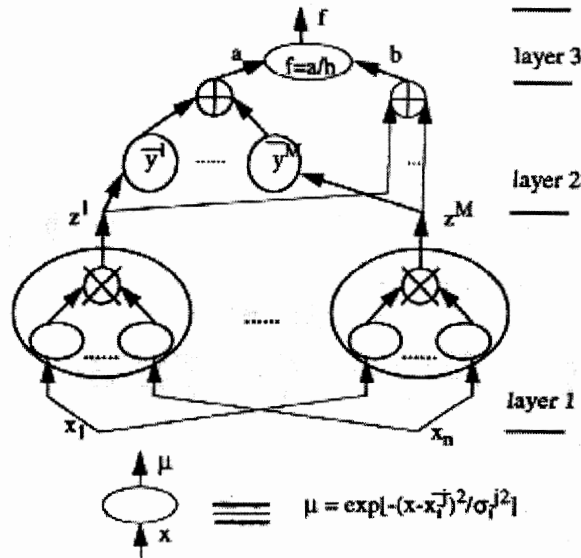


Figure 19: Network Representation of the Fuzzy Logic System
[4]

Please note that the notation with in Figure 19 is different from the notion which I am using in this report.

In my notation, $a = \sum_{l=1}^m u_l \mu_l$, $b = \sum_{l=1}^m \mu_l$ and $\mu_l = \prod_{i=1}^n \exp(-(\frac{x-a_i}{b_i})^2)$.

In Wang's work, he showed that given an input, output pair (x,d) , the above system would approximate a function $f(x)$ such that the error

8.
$$e = \frac{1}{2} [f(x) - d]^2$$

would be minimized. It is clear that the error is a cost function which is minimized using the gradient descent method. In our application, the square of the error does not represent a good cost function because we do not have a desired signal that we can compare with our output.

I chose to use a different cost function which is of the same form as equation (8) but does not compare signals. Instead, we are comparing the way point angle and the steering angle of the robot as seen in equation (9),

9.
$$J = \frac{1}{4} [WPA^2 + SA^2]^2$$

where WPA is the way point angle and SA is the steering angle. Perhaps I should redefine these variables for clarity. The way point angle is the relative angle between the robot and the desired destination point, see Figure 20.

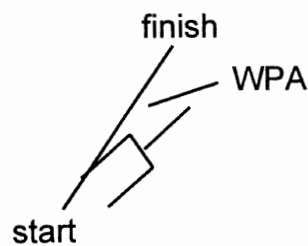


Figure 20: Definition of Way Point Angle

This makes it clear that if the way point angle and the steering angle are both zero, the cost function J in equation (9) will be zero. This only occurs when the robot is on the path which it had generated. Once it deviates from the desired path, equation (9) is no longer zero. In order to minimize equation (9), we must train the parameters u_1 , a_1^1 , and b_1^1 .

To train u_l we use

$$10. \quad u_l(k+1) = u_l(k) - \alpha \left. \frac{\partial J}{\partial u_l} \right|_k$$

where $l=1,2,\dots,m$, $k=0,1,2,\dots$, and α is a constant which controls the speed of convergence. It is clear that by using the chain rule, we have

$$11. \quad \frac{\partial J}{\partial u_l} = (WPA^2 + SA^2)SA \frac{\partial SA}{\partial a} \frac{\partial a}{\partial u_l} = (WPA^2 + SA^2)SA \mu_l \frac{1}{b}$$

where a and b are defined from Figure 19. We can now substitute (11) into (10) for the training algorithm for u_l :

$$12. \quad u_l(k+1) = u_l(k) - \alpha (WPA^2 + SA^2)SA \mu_l \frac{1}{\sum_{i=1}^m \mu_i}$$

where $l=1,2,\dots,m$, and $k=0,1,2,\dots$

To train a_i^l , we use

$$13. \quad a_i^l(k+1) = a_i^l(k) - \alpha \left. \frac{\partial J}{\partial a_i^l} \right|_k$$

and again apply the chain rule to get

$$14. \quad \frac{\partial J}{\partial a_i^l} = (WPA^2 + SA^2)SA \frac{\partial SA}{\partial \mu_i} \frac{\partial \mu_i}{\partial a_i^l} = (WPA^2 + SA^2)SA \frac{(\mu_l - SA)}{\sum_{i=1}^n \mu_i} \frac{2\mu_l(x_i - a_i^l(k))}{b_i^{l^2}(k)}$$

Substituting (14) into (13) we get the adaptation algorithm for a_i^l :

$$15. \quad a_i^l(k+1) = a_i^l(k) - \alpha (WPA^2 + SA^2)SA \frac{(\mu_l - SA)}{\sum_{i=1}^n \mu_i} \frac{2\mu_l(x_i - a_i^l(k))}{b_i^{l^2}(k)}$$

where $i=1,2,\dots,n$, $l=1,2,\dots,m$, and $k=0,1,2,\dots$

We can use the same method as shown above to determine the training algorithm for b_i^l :

$$16. \quad b_i^l(k+1) = b_i^l(k) - \alpha \left. \frac{\partial J}{\partial b_i^l} \right|_k = b_i^l(k) - \alpha (WPA^2 + SA^2)SA \frac{(\mu_l - SA)}{\sum_{i=1}^n \mu_i} \frac{2\mu_l(x_i - a_i^l(k))^2}{b_i^{l^3}(k)}$$

where $i=1,2,\dots,n$, $l=1,2,\dots,m$, and $k=0,1,2,\dots$

There is a two pass procedure to train the system shown in Figure 19. We first have to calculate forward along the fuzzy logic system to obtain μ^l and SA. Then the fuzzy logic system is trained backwards to obtain the new guidance control parameters u_l , a_l^l , and b_l^l by using the equations (12), (15), and (16).

Implementation of Adaptation Algorithm to AV Simulation

As was stated previously, the autonomous vehicle is intended as a test bed for new types of controllers developed in the future. If we are to keep this in mind, the adaptation algorithm must be modular so that it can be removed without altering the entire program. There are other considerations as well which are relevant to the cost function.

One such concern occurs when the AV reaches a way point and picks another in the desired path. It is possible to consider a condition where the second point is around some obstacle thus placing it on a separate path, see Figure 29 for an example. In these situations, for a brief time the cost function would be large. This situation is expected and we know it will happen quite often. If we were to allow the AV to run its adaptation routines during these points, it may be possible for the membership functions to diverge from the local minimum of the cost function surface. One possible solution is to only allow the AV to adapt if the way point angle is within a given range. I choose a range of plus or minus 45 degrees. This should guarantee that the AV is close to or on the desired path.

We also had to keep in mind how the way points are chosen within the program. Even if the AV does not know the environment, it will chose a way point. In this situation, the way point becomes the desired destination point. It is conceivable that this path would go through obstacles to which the robot is unaware. The robot would then use its obstacle avoidance routines to achieve that destination by going around these obstacles.

Then we must also observe how the obstacle avoidance rules coincide with the path planning routines. By looking at the code written by [1], we can see that the AV first determines the proper steering angle and

road speed needed to go in the direction of the desired way point. It will then obtain the data from the sensors and run through the obstacle avoidance rules. These rules will then alter the steering angle and robot speed in order to avoid a collision. This sequence of controllers gives the obstacle avoidance routine a higher priority over the path planning by overwriting the desired path.

So now we can see that even if we are operating at the local minimum of the cost function and we are on the path, the obstacle avoidance routines can force the AV to leave the path. If we were to allow the robot to always adapt its rules and we were forced off the path because of an obstacle, we would lose the local minima. It is impossible to remove this problem entirely as will be shown in the results and conclusions. However, we can make its effects less evident by only allowing the AV to adapt when it knows the path to the desired location. By limiting the adaptation to known paths, we still have to contend with obstacles, but the AV generally knows where it is going and where the obstacles are located (excluding moving or new obstacles).

The final implementation consists of two conditions before the AV is allowed to adapt its membership functions. The first is a flag is either true or false. The flag is made true any time a path is generated within a known environment. Once the flag is made true, the AV then checks the way point angle. If it is within the allowed plus or minus 45 degrees, the robot is allowed to adapt its rules.

CHAPTER VII

SIMULATION RESULTS AND DISCUSSION

Non Performance Results

There are now three different versions of the simulation program. The first was written by [1] and contains the trapezoidal membership functions and the fuzzy logic library written by [1]. This version (from this point on referred to as RA) was modified to the new mapping algorithm so that all three simulations could be compared, but this is the only change. The second version (from this point on referred to as FLC) uses the Gaussian membership functions, Larsen's rule and max-prod composition rule. This version does however, use the same rules as written by [1] as stated previously in this report. The third and final version of the simulation (now referred to as Adapt) uses the same rules as the FLC version with the addition of the adaptation routines.

There are two considerations concerning nonperformance results, the size of the executable program and the time required for the computer to run through the supervisor routines. While computer memory is becoming less and less expensive, the required memory could still be a concern when using older CPU's. Table 1 compares the size of each executable in bytes.

Table 1: Comparison of Executable Memory Requirements

Simulation	size (bytes)
RA	156478
FLC	150055
Adapt	150729

It is clearly evident that the new fuzzy inference engine saves computer memory. This is because the new inference engine can be reduced to one equation (7) where as the inference engine used by [1] required

several pages of computer code. The Adapt version required slightly more memory due to the adaptation routines.

We can also compare the time required to run through the fuzzy logic routines themselves. Since the only difference between the separate versions is the fuzzy logic rules, it does not make sense to time the whole simulation at this time. By using the time function in [8], it was possible to determine the actual computing time. Since I only want to compare the time savings between the different inference engines, the Adapt time is irrelevant. Table 2 compares these times.

Table 2: Time Required to Compute Fuzzy Logic Output

Simulation	Average Time (sec)
RA	0.0756
FLC	0.0625

The time was calculated by running through the routines one thousand times and computing the average on a 486DX40 computer with a math coprocessor. The reader can see that the FLC version is about 17% faster than the RA version. This time savings could be important in the future if voice commands or some other types of improvements are made.

Performance Results

The performance results will compose of four different comparisons. Since the program only adapted in known environments, data was only collected if a known path was generated. The first comparison was the way point angle. This measure will show how well the vehicle stayed on the path. As stated previously in this report, a way point angle of zero signifies the robot is on the path.

The second comparison is the steering angle as commanded from the fuzzy logic rules. This will be a good indication of the smoothness of the ride. The smoother the curve, the smoother the ride. This is an

important consideration for the wheelchair application. Nobody would purchase a wheelchair which had a ride similar to that of a roller costar.

The third comparison is the cost function itself. Since the adaptation routines are trying to minimize the cost function using a gradient descent method, we can observe how fast equation (9) goes to zero as compared with the non adaptive fuzzy controllers. To evaluate this comparison, an extra line had to be added to the code of the non adaptive systems to determine the cost function. Even without the adaptive routines, this will still give a good performance measure because it should remain zero if the car is on the path.

The forth and final comparison is the amount of time the simulation took to reach the desired destination. The time can be read from each of the plots on the x-axis. While a fast time is not necessarily a good thing (we don't want a wheelchair going 30 mph) it does show how efficient the code is at providing the proper commands.

Before I get into the different trials performed, I would like to describe how I will discuss them. Each trial will first be shown with a photo of the screen which will show the car, the desired destination, and the determined path. Each trail was picked in order to give a range of possible situations which the AV may encounter. I will then show each of the three plots of way point angle, steering angle, and cost function. The notation used for the three controllers in the last section will also be used on the plots. I will then discuss my results for each case and provide a general result for the project in the next chapter.

Trial #1

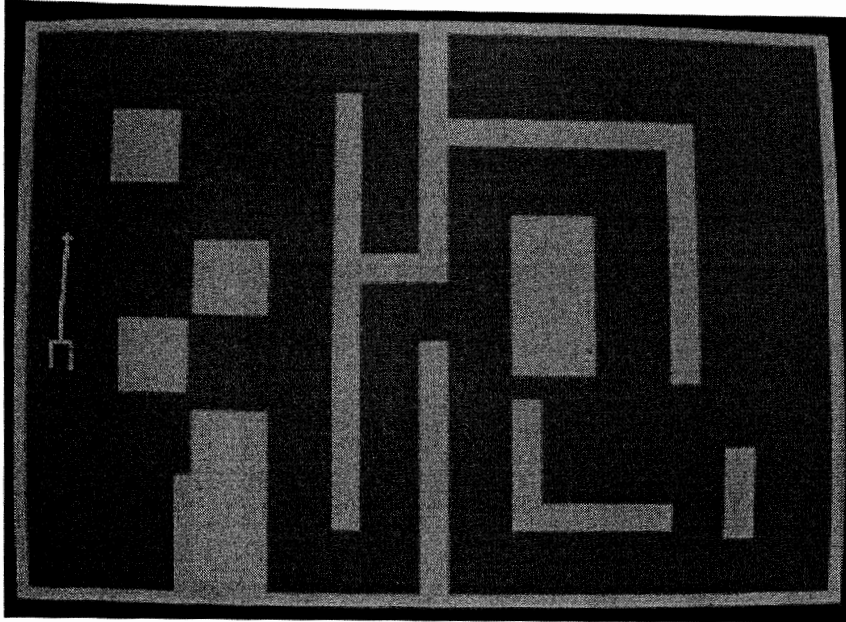


Figure 21: Generated Path for Trial One

The first trial's objective is to travel a straight line to the target a few meters away. It should be noted that there is a little distortion in all of the screen photos due to the curvature of the computer monitor and the scanner's resolution. I would also like to note that the maze environment was created by [1] with the white areas signifying an obstacle or wall, and the gray areas signifying free space.

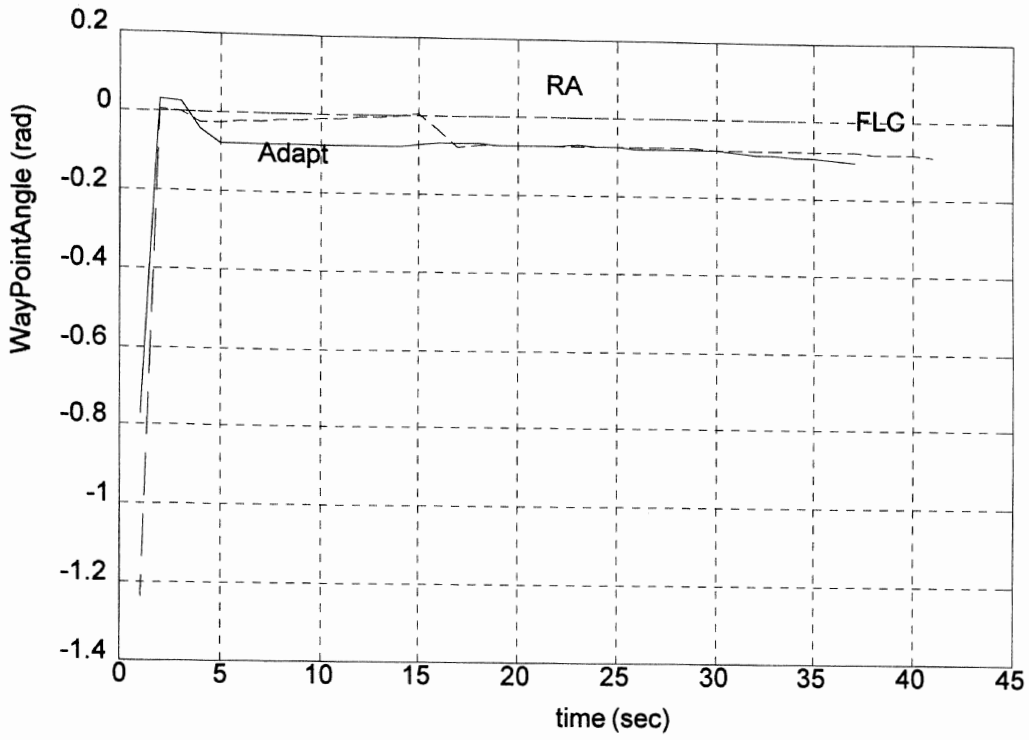


Figure 22: Way Point Angle for Trial #1

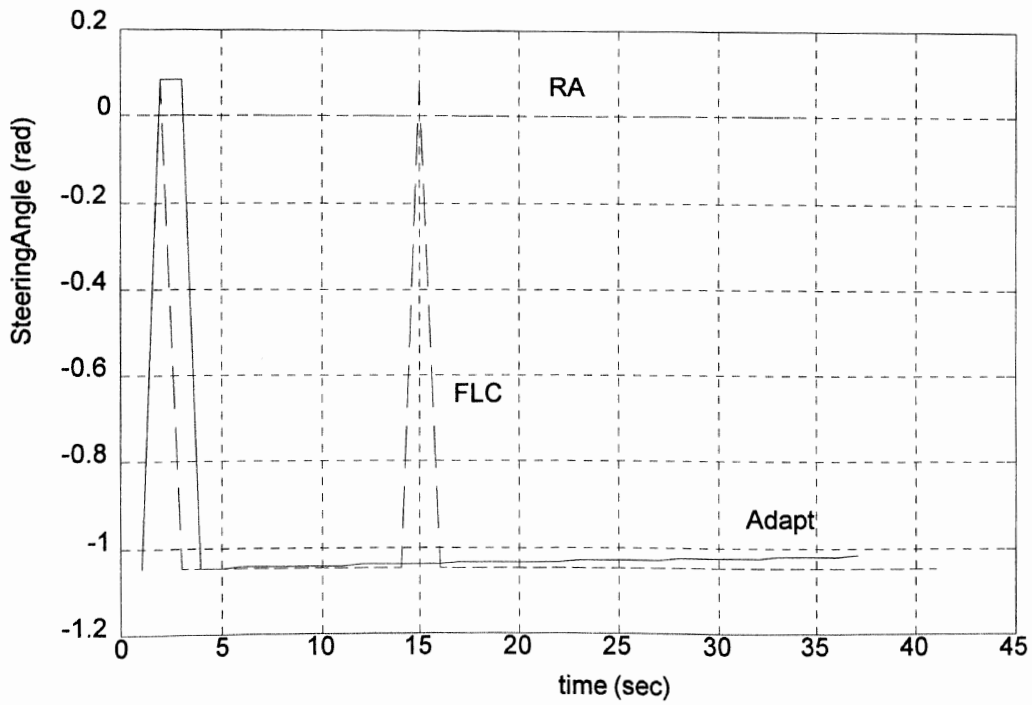


Figure 23: Steering Angle for Trial #1

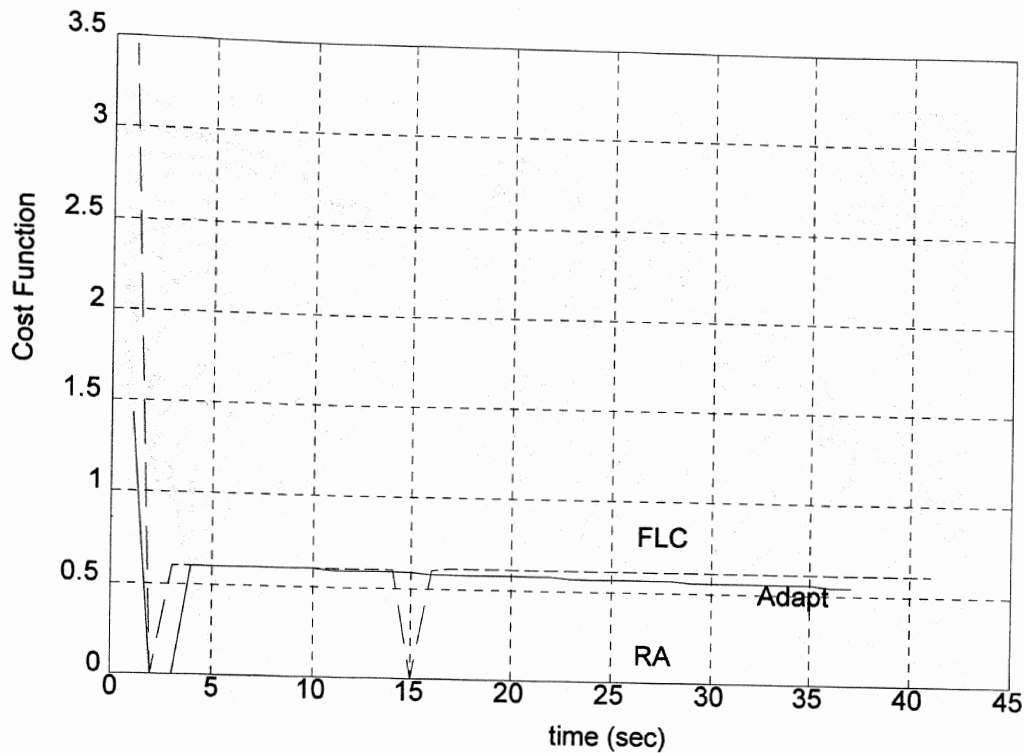


Figure 24: Cost Function for Trial #1

Trial #1, as will be shown in trials 2 through 5, is the only case where RA has an outright advantage. When the program is started, FLC and Adapt tend to drift around the starting point while RA holds its position. This is due to the fact that Gaussian membership functions are always true to some degree, even if quite small. The dynamic model used for the simulation does not include friction thus the small truth generated will cause the AV to drift. It is believed that the friction inherent in the robot will be great enough that the implemented system will not drift.

Note that the spike in the steering angle for the FLC case is because the obstacle avoidance routine caused the AV to deviate slightly. This will be seen in almost every trial.

As for the conclusions for this case, RA not only has a zero cost function, steering angle, and way point angle, it also finished the task in shorter time. The Adapt routine has the next best response because it has a smoother ride and takes less time than the FLC routine.

Trial #2

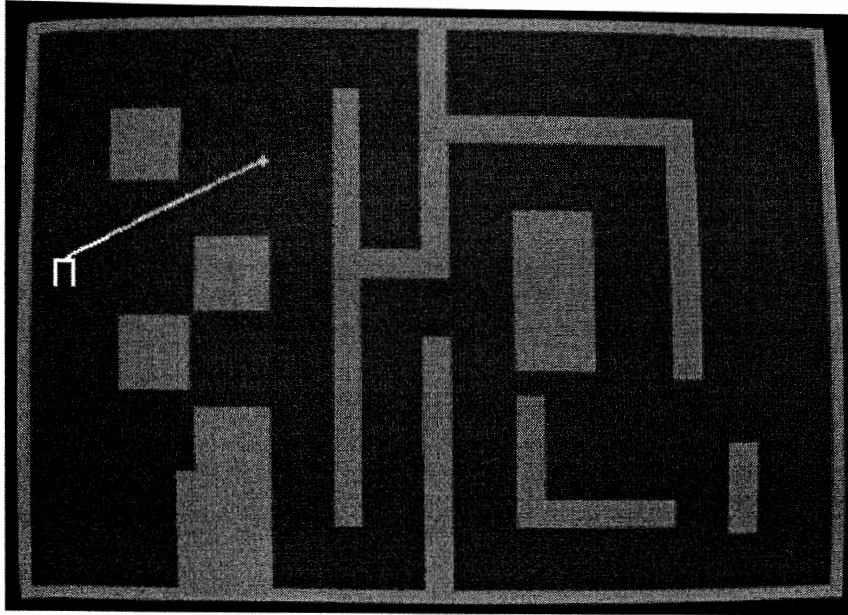


Figure 25: Generated Path for Trial Two

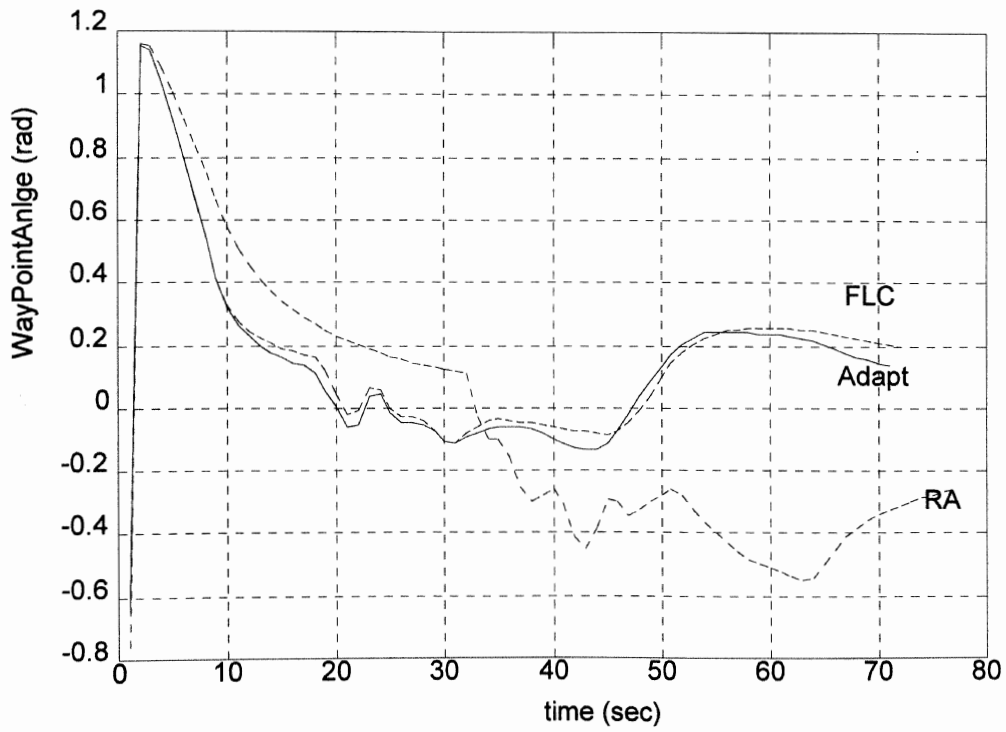


Figure 26: Way Point Angle for Trial #2

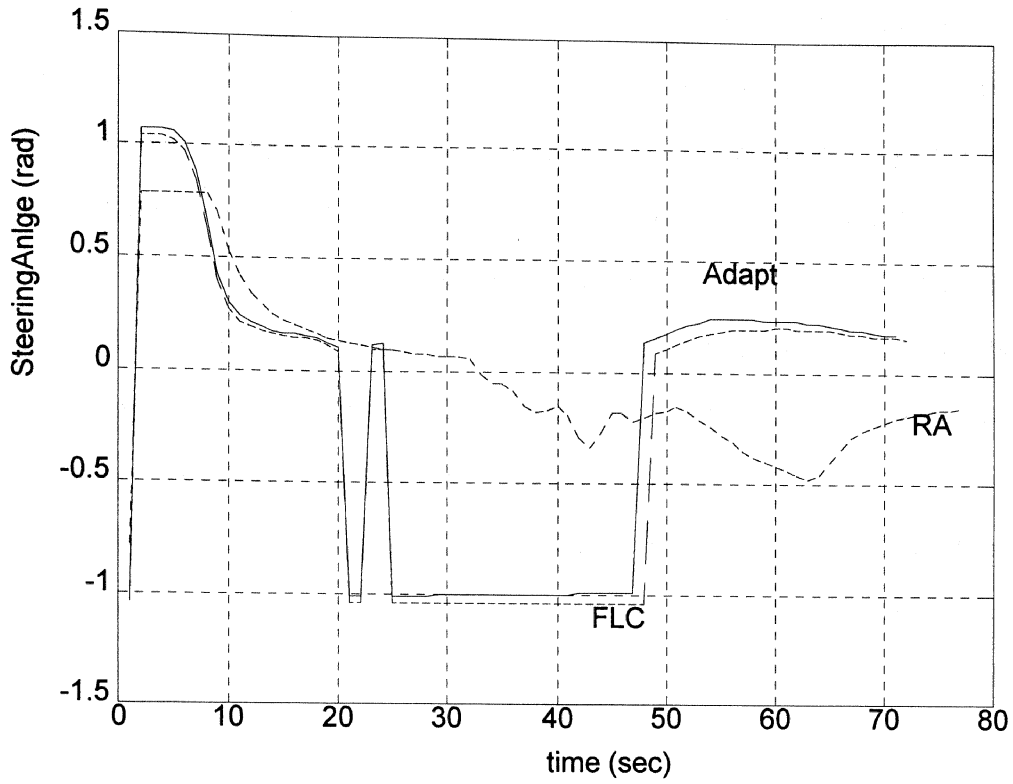


Figure 27: Steering Angle for Trial #2

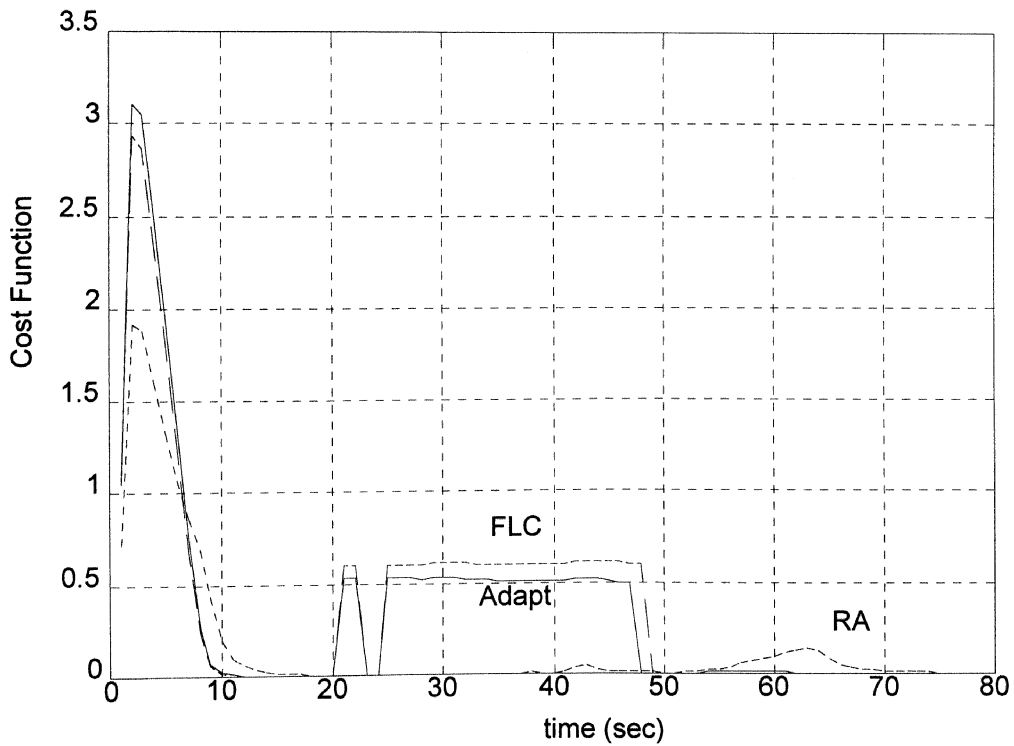


Figure 28: Cost Function for Trial #2

Trial #2 has the AV going through a turn to the right and proceeding through two obstacles to the desired location as shown in Figure 25. By looking at the plots for WPA, SA, and J (cost function), we can see that after the turn is completed, at around $t=3$ sec, where all three simulations begin to converge on the path. At about 20 sec, the robots begin to encounter the obstacles which must be avoided.

In Figure 26, the way point angle is bounded within plus or minus 0.2 rad in the Adapt and FLC simulations. This is implying that these controllers held the path with greater accuracy than that of the RA simulation. However, Figure 27 shows that the steering angle for the Adapt and FLC cases has much sharper changes in the steering angle. This would produce a jerky ride.

The cost function shown in Figure 28 shows that the Adapt and FLC cases converges to zero faster but increases around the obstacles which is expected. The unexpected result is that the RA case did not increase as to the same levels. This is due to the smoother ride because the SA did not jump for this simulation. However, once past the obstacles, the FLC and Adapt cases went back to zero and held a zero cost function while the RA case jumped.

There is also a time reduction in the Adapt and FLC cases over the RA case. While not a substantial time reduction, the Gaussian functions do achieve the desired destination at a quicker rate.

In the case of trial two, the Gaussian membership functions achieved better results. We don't see a substantial difference between the Adapt and FLC cases, but the Adapt does hold the path slightly better than that of the FLC simulation.

Trial #3

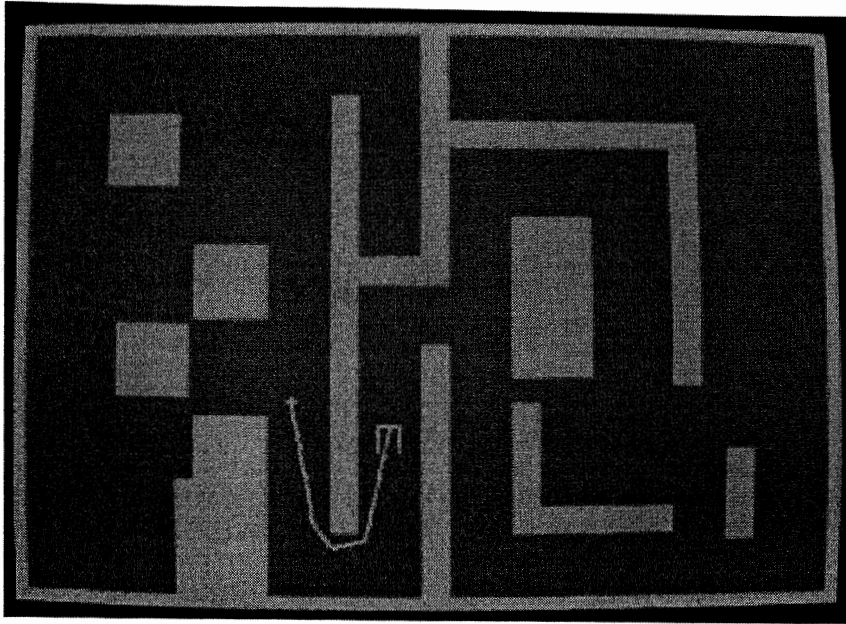


Figure 29: Generated Path for Trial Three

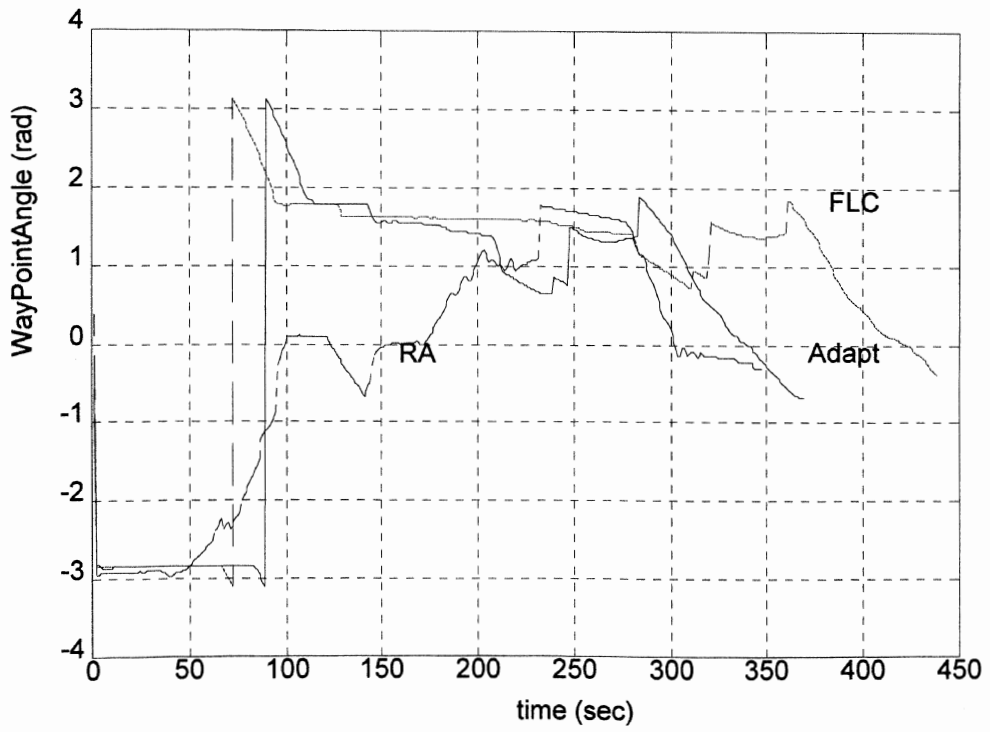


Figure 30: Way Point Angle for Trial #3

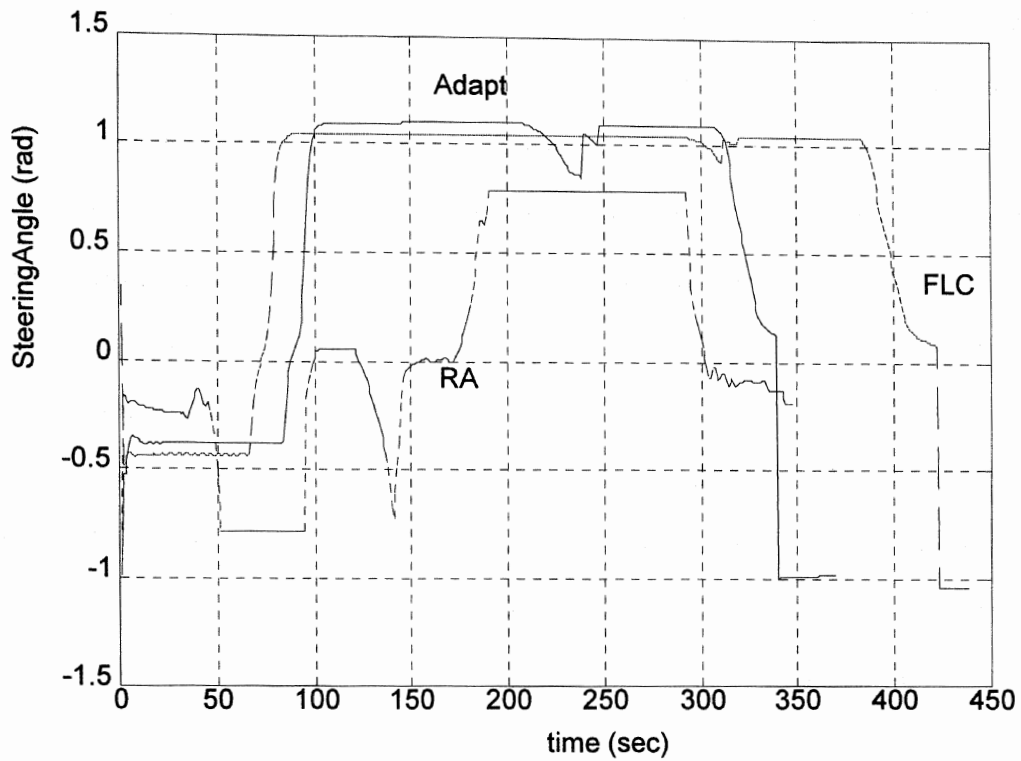


Figure 31: Steering Angle for Trial #3

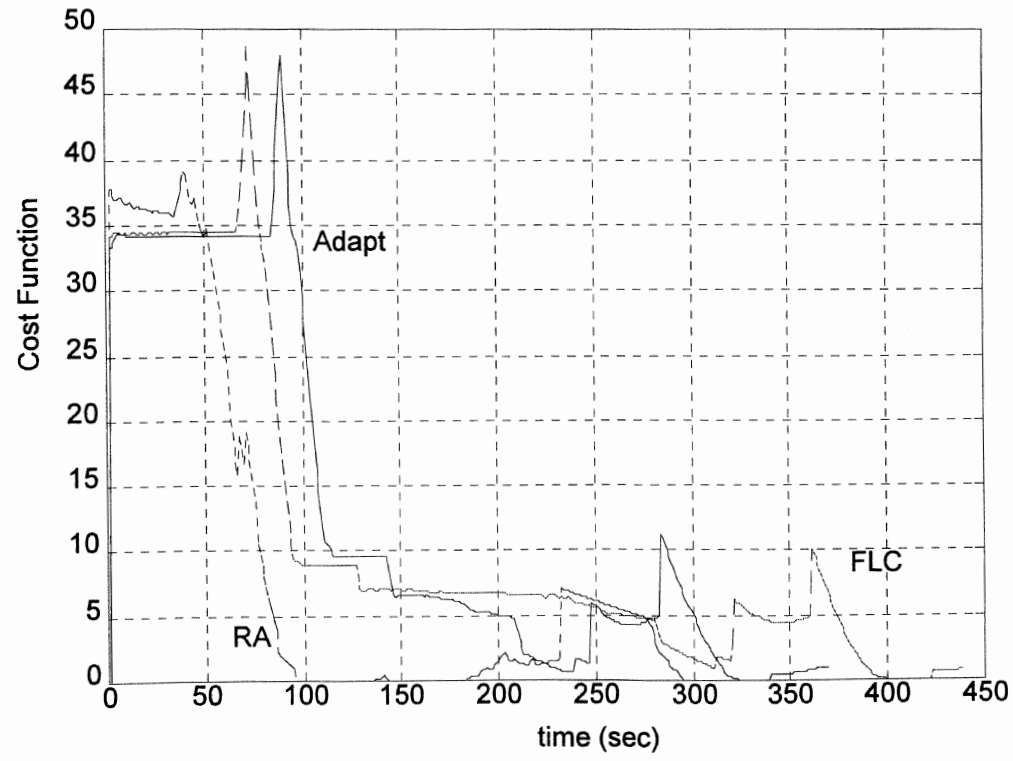


Figure 32: Cost Function for Trial #3

Trial #3 has an interesting twist to staying on the path. At the starting time, the AV is pointing in the wrong direction as shown in Figure 29. The programs behaved in slightly different manners through this maneuver. For instance, the RA simulation went in reverse until it could make the turn. The Adapt and FLC simulations both turned within the hall way. This is a very difficult maneuver and took additional time over the RA case. However, both cases did successfully turn around within the hall way to continue on the path. Another interesting condition in this trial is the turn back towards the north (top of the screen). All three cases achieved this maneuver without difficulty.

Looking at Figure 30, we can identify several of the above maneuvers. The horizontal portions at the beginning of the simulation are where the vehicles were either turning around or following the path in reverse. At about $t=200$, the AV is making the turn around the wall. Another interesting observation is the instance when the obstacle avoidance routines have corrupted the path following commands. This is seen with the jagged edges near the end of the trail.

While the RA simulation does minimize the cost function better than the Adapt and FLC trails, the Adapt and FLC trials produced a smoother ride. The time to complete the task is hard to compare as they did not solve it in the same manner. It took more time to turn the AV around thus causing these trials to take more time. However, the Adapt simulation was not considerably slower and was comparable to the RA case.

One observation which may have been noticed by the reader is that on some of the trails, the cost function never reaches zero or the way point angle and steering angles end at some finite value. This occurs when the robot reaches the desired destination at a skewed angle. The simulations stop when the AV gets within a certain distance of the destination. It does not require the way point angle to be zero to complete its mission.

Trial #4

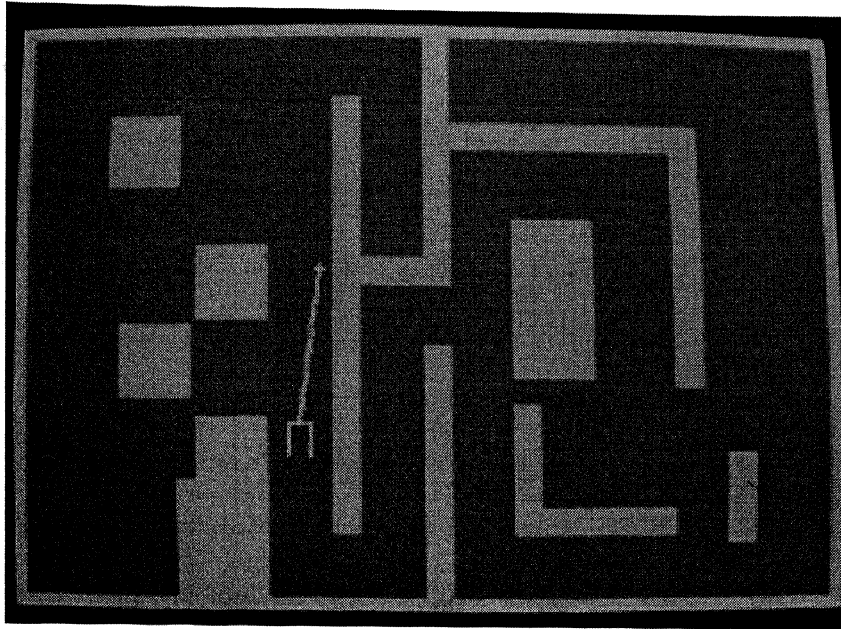


Figure 33: Generated Path for Trial Four

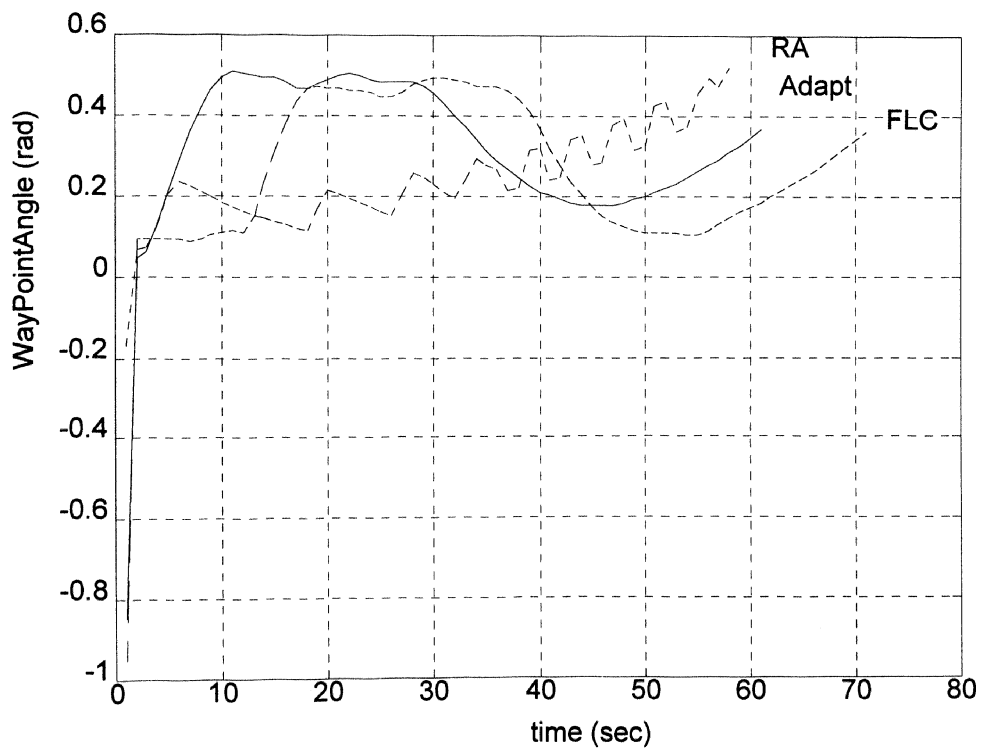


Figure 34: Way Point Angle for Trial #4

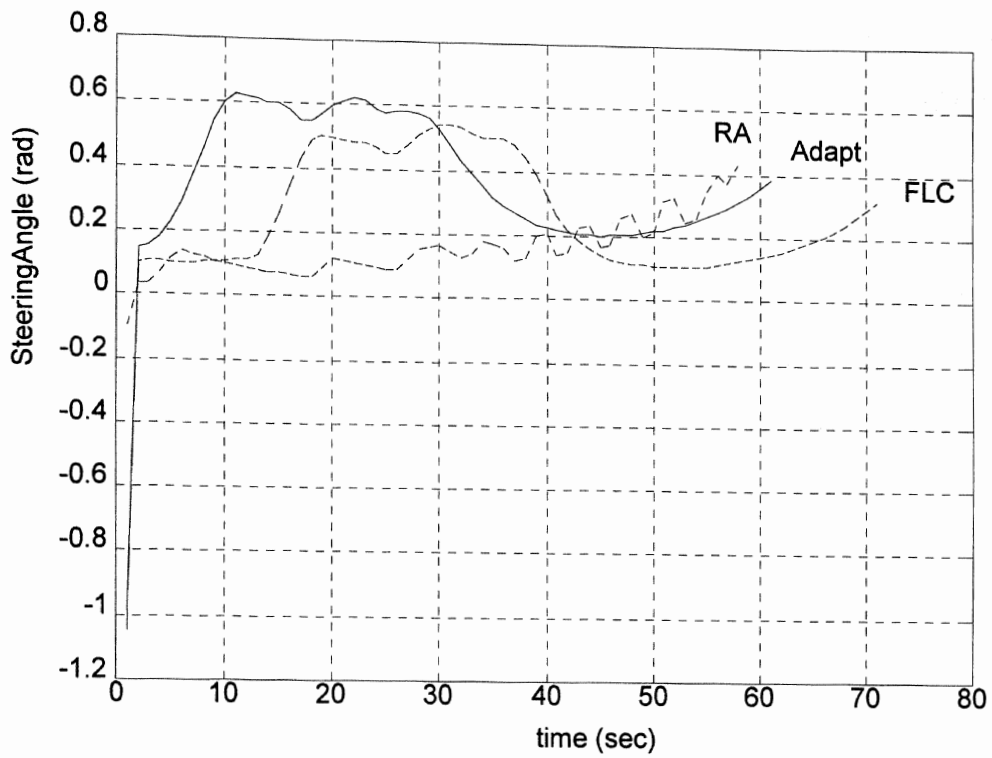


Figure 35: Steering Angle for Trial #4

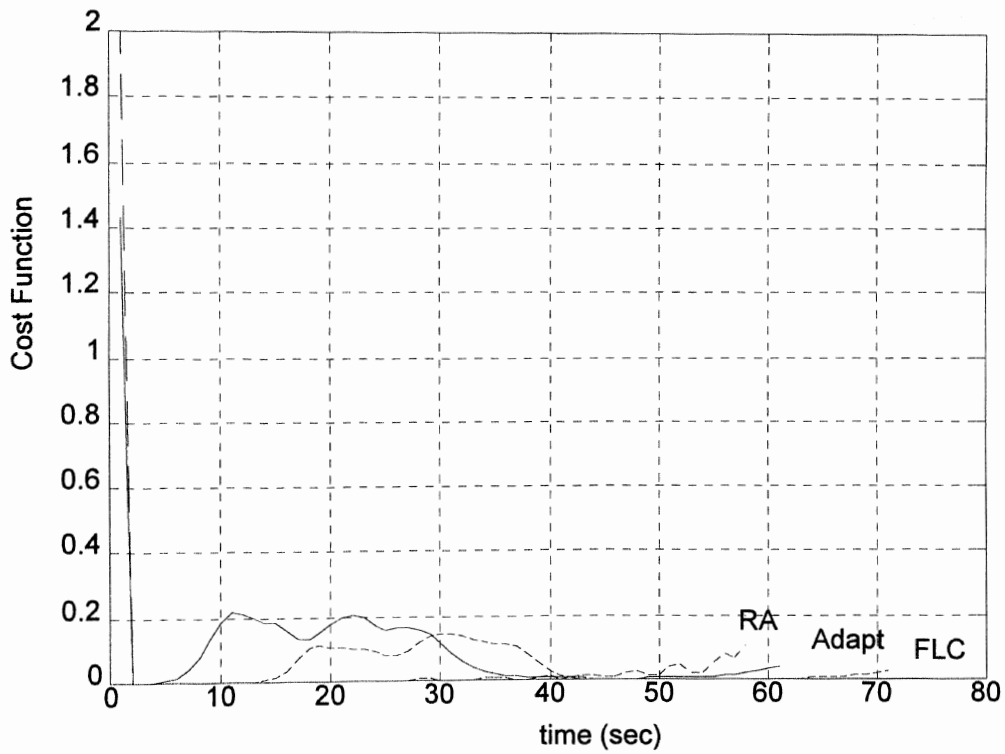


Figure 36: Cost Function for Trial #4

Trial number four is simulating the condition where the robot has to go down a hallway but towards the right wall. This becomes an important simulation when the robot will probably be used inside a building where hallways are encountered quite often.

Looking at Figures 34 through 36 we see a spike in the first few seconds. This is due to the offset from not hitting the previous obstacle with zero heading angle. We can then identify the hump as when the AV reaches the point with the hole in the wall. At this point, the obstacle avoidance sends the robot to the left in order to avoid the wall on the right. The fuzzy rules do eventually bring the robot back on track to reach the destination point. Again we see the offset at the end of the simulation. This is due to the avoidance rules turning the AV away from the wall when it reaches the final location.

The time required to accomplish the mission is relatively the same between the RA and Adapt simulations while the FLC case took much longer. The Adapt case minimizes J faster which means it is able to hold the path through the task. The noticeable difference in this trial is the oscillation in the RA simulation. This is caused by the obstacle avoidance rules trying to get away from both the right and left walls of the hallway. This is very undesirable in an environment where people are walking down the same hallway as the robot. The Adapt and FLC simulations were able to smooth out the ride giving a better response.

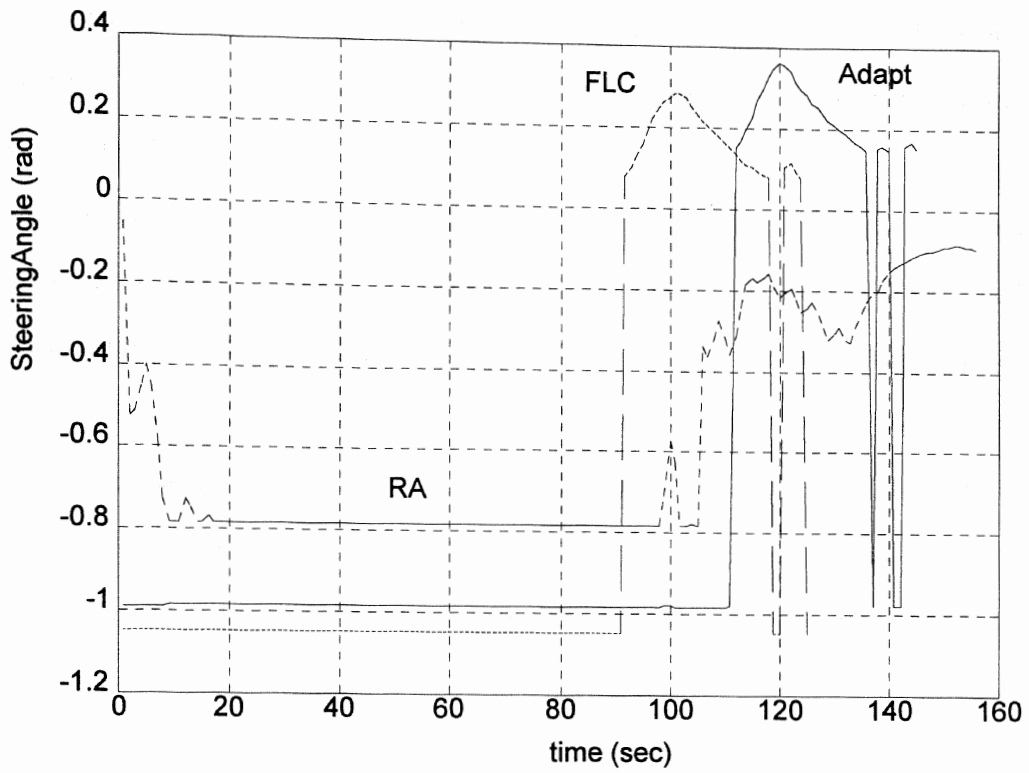


Figure 39: Steering Angle for Trial #5

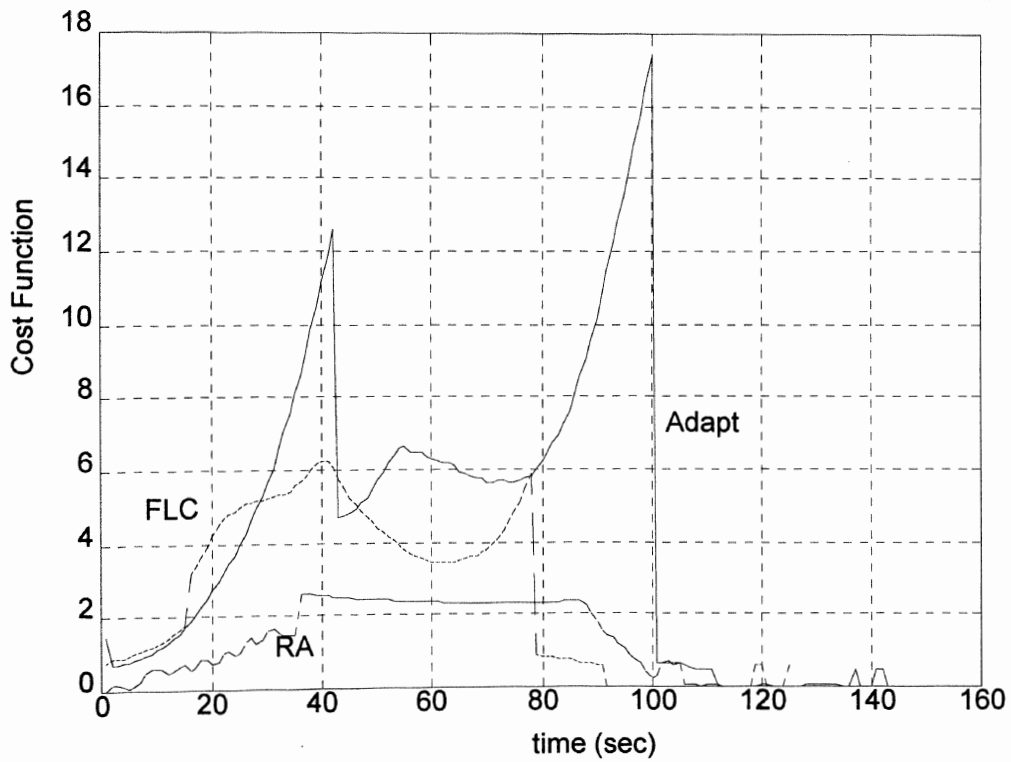


Figure 40: Cost Function for Trial #5

Trial number five is another common situation which may be encountered within a building environment. This situation is analogous to the robot moving into a room and having to go around and obstacle near the door.

Initially, the way point angle was off quite a bit due to the location of the desired target. By looking at Figure 38, we can see the location of the obstacle is at about 40 sec. One observation between the different controllers is the amount of overshoot around this obstacle. The RA simulation hugged the obstacle a lot closer than either the Adapt or FLC simulations. One unexpected result in this trial was the FLC achieving the final destination at a much quicker pace than either of the other controllers. There was quite a bit of adaptation going on within the Adapt routine because of the shape of the target path. This path was generally within the plus or minus 45 degrees most of the time. This would account for the extra time in the Adapt routine.

We can also see that time less than 115 seconds, the Adapt and FLC routines are much smoother than the RA simulation. However, when the robot gets near the final destination, the Adapt and FLC both oscillate their steering angles with large deflections. The most likely reason is that the robot is heading almost straight into the wall on the left of the room. This would cause the obstacle avoidance routines to try and turn while the path following rules are trying to go straight. This problem will always occur as long as the obstacle avoidance routines have higher priority over the path.

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

This project consisted of three separate parts. The first was implementing the adaptive mapping routines written by [1] into a format which could be used by the robot. The idea was to keep the same format as developed by [1] but use computer memory instead of the computer's monitor. The final functions were written in a portable manner so that in the future, if a new map is generated, all the user will have to do is swap functions.

The second part of the project was to implement the Gaussian membership functions with fuzzy logic max-prod and Larsen's rule for composition. This was done in order to save computer time and memory as well as make for an easier transition into the adaptive fuzzy logic controller. In addition, the Gaussian membership functions provide for nice exponential output which is well understood.

The third and final aspect of this thesis was to add an adaptive routine into the Gaussian membership functions. Again we wanted to keep the routines portable so different types of controllers could be placed within the simulation. We had to remember that the purpose of the test bed is to test new complex real time controllers. Keeping the controllers portable makes it easier to change and try new ideas.

The author's contributions to this project are,

- Implementation of the Bit Field Map Concept developed by [1].
- Developed a second fuzzy logic controller using the same rules as [1] but with a different inference engine.
- Developed an appropriate cost function to be used with the gradient descent algorithm for the fuzzy logic adaptation routines.
- Implemented the third new controller composing of a back-propagated adaptation algorithm to adapt the fuzzy logic membership functions.

The simulations were all executed on an 80486DX40 based PC. The results showed that different controllers worked better in different situations. Generally speaking, the Adaptive and Gaussian membership functions outperformed the original simulation written by [1]. The problem with using this

type of system is the conflict which exists between the two controllers (path following and obstacle avoidance). As long as the two controllers work against each other, there will be inconsistencies between the different situations. By adding more rules or even changing the architecture of the rules to one controller, some of these problems could be eliminated. However, the idea of the obstacle avoidance routines having a higher priority over the path following rules is desirable to avoid any accidents, especially when the environment involves human interaction.

Future Work

At this time, the hardware for the wheelchair is being assembled and should be ready for testing by the end of the summer. The proposed implementation [1] using the CAN standard is being used with 68HC11 micro controllers and one 486DX40 PC board. The use of the PC Board will provide the team with a lot of space for expansion of the robot's abilities. In the future, human interaction will have to be developed as we are currently using keyboards. A voice command interface would provide for good interaction with the user. As stated by [1], path planning for multiple targets will be devised and implemented. The possibility also exists for trying different types of vision such as inferred or pattern recognition. The autonomous vehicle will be used as a test bed for new control methodologies for distributed real time control problems.

References

- [1] Andujar, R. "*Autonomous Vehicle Control Using Fuzzy Inference and a Fast Path Planning Algorithm.*" M.S. Thesis, Department of Mechanical and Aerospace Engineering, Oklahoma State University, 1991.
- [2] The Truck & Bus Control and Communications Network Subcommittee of the Truck & Bus Electrical & Electronics Committee, Society of Agricultural Engineers. Recommended Practice for Serial Control and Communications Network (Class C) for Truck and Bus Applications. In SAE J1939, SAE Publications, 1993.
- [3] Wang, Li-Xin, "*Analysis and Design of Fuzzy Systems,*" USC SIPI Report, No. 206, 1992.
- [4] Wang, Li-Xin, "*Adaptive Fuzzy Systems and Control, Design, and Stability Analysis,*" Prentice-Hall, Inc., 1994.
- [5] Lee, Chun - Lin, "*Ultrasonic Ranging System of an Obstacle - Avoidance Robot,*" M.S. Thesis, Department of Mechanical and Aerospace Engineering, Oklahoma State University, 1993.
- [6] Newton, John, Correspondence concerning current research. Department of Mechanical and Aerospace Engineering, Oklahoma State University, 1994.
- [7] D'Offay, Philippe, MAE 4010 report, Oklahoma State University, Spring 1994.
- [8] Borland C++ reference manuals v. 3.1, copyright by Borland International, INC. Scotts Valley CA 95067-0001.
- [9] Alberto, Pedro, "*Fuzzy Logic Controllers, A design Methodology,*" Notes from presentation made at Oklahoma State University, 1993.
- [10] Farrell, J.A., and Baker, W.L. "*An Introduction to Learning Control Systems.*" 8th IEEE Int. Symp. on Intelligent Control, 25 Aug. 1993.
- [11] Wang, B.H. and Vachtsevanos, G. "*Learning Fuzzy Logic Control: An Indirect Control Approach,*" IEEE Int. Conference on Fuzzy Systems, March 1992.
- [12] Driankov, Dimiter, "*An Introduction to Fuzzy Control,*" Springer-Verlag, 1994.
- [13] Zimmermann, H. "*Fuzzy Set Theory and its Applications,*" Kluwer Academic Publishers, Norwell, Massachusetts, 2 ed. 1991.

APPENDIX A: MAP SIMULATION SOFTWARE CODE AND DOCUMENTATION

GLOBAL VARIABLES

Variable	Description	Units
theta	The half arc angle of the sonars.	radians
map[][]	Array containing the environmental bit field map.	integers
zero	x=0, y=0 in the Cartesian plane of the map.	bits
resolution	The resolution of the map.	m/bit
s_byte	The size of a byte.	bits
max_dist	The maximum distance of the sonar.	meters
ones	The equivalent of having an integer with all bits a '1'	

STRUCTURES

The only structure in this program is called bit. It creates a variable which stores the row and column bit of any location in the map. For instance, bit zero{320,240} says that the zero location in the map is at column bit 320 and row bit 240.

FUNCTIONS

int main(void)

This is the main function of the map simulation. The first thing it does is initialize the graphics using *int_graphics()*. It then clears the map making all space occupied and prints it to the screen using *print_map()*.

The function then accepts the following data from the user in the following order:
current location of robot x y in Cartesian coordinates.
the distance return by the sonar (distance to obstacle) in meters.
the heading angle of the sonar in radians.

Note that the simulation will not prompt the user for this information, he just types it in.

Once the data has been entered, the triangle is defined by finding the three points of the triangle and the slope and y-intercept of each line between the three points. Note, the units of all these variables is still in the Cartesian plane in SI units.

The function then runs a loop for each row of bits in the triangle by finding the left and right bits for that row. This is done using *xy_to_bit(x,y)*. Once a row of bits has been defined, it clears that row using *clear_row(l_bit,r_bit,...)* then proceeds to the next row. After the triangle has been cleared, it makes corrections for any obstacles which may have moved into the area and then prints out the map.

```

/*****
* Simulation program to test bit map creation and alteration
* R. Shanley III
*
* last modified: 16Mar94
*
*****/

```

```

#include<stdio.h>
#include<math.h>
#include <graphics.h>
#include <stdlib.h>
#include <conio.h>

```

```

/* 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;

```

```

void init_graphics(void);
void print_map(void);
bit xy_to_bit(float, float);
void clear_row(bit, bit,int,int);

```

```

double theta=.209440; /* arc of sensor in radians */
unsigned int map[480][40]; /* map[row][column] -- the actual map*/
bit zero={320,240}; /* (x,y) origin in the bit field */
double resolution=.1524; /* map resolution in m/bit */
int s_byte=sizeof(unsigned int)*8; /* the size of a byte in bits */
float max_dist=10.0; /*maximum distance of sonar in meters */
int maxx,maxy,ones;
int main()

```

```

{
    int row,column,r,i;
    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;

```

```

/* initialize the graphics. For simulation only */

```

```

    init_graphics();

```

```

/* 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);

    for(row=0;row<=maxy-1;++row)
        for(column=0;column<=maxx-1;++column)
            map[row][column]=ones;

/* print out the map to the screen
 * --used only in this simulation
 */

    print_map();

/* start the simulation forever--used only in
 * simulation as the supervisor will replace
 * this section of code. The idea is to have
 * stand alone functions which will alter the
 * map.
 */

    for(;;)
    {

/* enter the sensor data--will be replaced
 * with data delivered from the CAN
 */

/* 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.
 */
        r=0;
        scanf("%lf %lf %lf %lf",&x,&y,&d,&phi);
        if (d >= max_dist)
            r=1;

/* 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 < 0.0)
            phi=6.28318+phi;

```

```

else if (phi > 6.28318)
    phi=phi-6.28318;
if (phi > 4.71 && phi < 4.72)
    phi=4.710;

w=d/cos(theta);

/* w is length of triangle sides */

/* If heading angle is in the top half of the
* plane, define the triangle as following:
*/

if ((phi >= 0 && phi <= 1.57) || (phi > 4.71))
{
    a=x+w*cos(1.57-theta-phi);
    b=y+w*sin(1.57-theta-phi);
    if (b > -.01 && b < .01)
        b=y;
    p=x+w*sin(phi-theta);
    q=y+w*cos(phi-theta);
    if(q > -0.01 && q < .01)
        q=0.0;
    m1=(b-y)/(a-x);
    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;
    m3=(q-b)/(p-a);
    if(m3 > -0.01 && m3 < 0.01)
        m3=0.0;
    t1=y-x*m1;
    t2=y-x*m2;
    t3=q-p*m3;
}

/* (a,b) are coordinates of */
/* right point on triangle */
/* round to zero if small */

/* (p,q) are coordinates of */
/* left point on triangle */
/* round to zero if small*/

/* slope of line 1 (x,y) (a,b) */
/* round to zero if small*/

/* slope of line 2 (x,y) (p,q) */
/* round to zero if small*/

/* slope of line 3 (a,b) (p,q) */
/* round to zero if small*/

/* y-intercept for line 1 */
/* y-intercept for line 2 */
/* y-intercept for line 3 */

/* If heading angle is in the bottom half of the
* plane, define the triangle as following:
*/

else if (phi > 1.57 && phi <= 4.71)
{
    a=x+w*sin(phi-theta);
    b=y+w*cos(phi-theta);
    if (b < .01 && b > -.01)
        b=y;
    p=x+w*cos(1.57-theta-phi);
    q=y+w*sin(1.57-theta-phi);
    if(q < .01 && q > -.01)
        q=0.0;
    m1=(b-y)/(a-x);
    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;
        m3=(q-b)/(p-a);
        if(m3 > -0.01 && m3 < 0.01)
            m3=0.0;
        t1=y-x*m1;
        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 ((phi >= 0 && phi <= 1.57-theta) || (phi > 4.71+theta))
    {

/* The following are when the cone has one
 * side horizontal or slope of zero.
 */
        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)
            /* generate the incremental bit */
            /* increment the incremental bit */
            /* 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;
        tmp_x=x;
        /*the line of bits currently correcting */

/* 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 obstacle is present. 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 position
 * compared 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=floor((l_bit.col_bit)/s_byte);
        e=l_bit.row_bit;
    }

```

```

        f=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 ((phi > 1.57+theta) && (phi <= 4.71-theta))
{
    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 < 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);
        if (tmp_y < q && m3 != 0)
        {
            t=2;
            tmp_x=(tmp_y-t3)/m3;
        }
    }
}

```



```

        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 orientation of the triangle and
* specifies the line to which the next
* row originates.
*/
        if (phi > 1.57)
        {
            if (m1 == 0)
                tmp_x=tmp_x+resolution;
            else
                tmp_x=(tmp_y-t1)/m1;
        }
        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 to next row */
    tmp_y=y-resolution;
    if (phi > 1.57)
    {
        if (m2 == 0)
            tmp_x=tmp_x+resolution;
        else
            tmp_x=(tmp_y-t2)/m2;
    }
    else
    {
        if (m1 == 0)
            tmp_x=tmp_x+resolution;
        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 > 1.57)
    {
        if (m2 == 0)
            tmp_x=tmp_x+resolution;
        else
            tmp_x=(tmp_y-t2)/m2;
    }
    else
    {
        if (m1 == 0)
            tmp_x=tmp_x+resolution;
        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 (phi > 3.14159)
{

    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 < 4.72)
        {
            if (m2 == 0)
                tmp_x=tmp_x+resolution;
            else
                tmp_x=(tmp_y-t2)/m2;
        }
        else
    }
```

```

        {
            if (m1 == 0)
                tmp_x=tmp_x+resolution;
            else
                tmp_x=(tmp_y-t1)/m1;
        }
        tmp_bit=xy_to_bit(tmp_x,tmp_y);
    }
    tmp_y=y-resolution;
    if (phi > 4.71)
    {
        if (m2 == 0)
            tmp_x=tmp_x+resolution;
        else
            tmp_x=(tmp_y-t2)/m2;
    }
    else
    {
        if (m1 == 0)
            tmp_x=tmp_x+resolution;
        else
            tmp_x=(tmp_y-t1)/m1;
    }

    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 > 4.71)
        {
            if (m2 == 0)
                tmp_x=tmp_x+resolution;
            else
                tmp_x=(tmp_y-t2)/m2;
        }
        else
        {
            if (m1 == 0)
                tmp_x=tmp_x+resolution;
            else
                tmp_x=(tmp_y-t1)/m1;
        }
        tmp_bit=xy_to_bit(tmp_x,tmp_y);
    }
}
}

```

```

/* print map out -- simulation only */
print_map();

```

```

    }
}
/* initializes the graphics routines */

void init_graphics(void)
{
    int gdriver = DETECT, gmode, errorcode;

    initgraph(&gdriver,&gmode,"");
    errorcode=graphresult();
    if (errorcode != grOk)
    {
        printf("\nGraphics error %s",grapherrormsg(errorcode));
        printf("\nPress any key to halt:");
        getch();
        exit(1);
    }
    maxx=getmaxx() + 1;
    maxy=getmaxy() + 1;
    graphdefaults();
    cleardevice();
    return;
}

/*****
 * prints the map out on the screen
 *****/

void print_map()
{
    int i,mask=1,row,column;
    unsigned int tmp;
    for (row=140;row<=340;++row)
    {
        for (column=15;column<=25;++column)
        {
            tmp=map[row][column];
            for (i=0;i<s_byte;++i)
            {
                if(tmp & mask)
                    putpixel(column*s_byte+s_byte-1-i,row,2);
                else
                    putpixel(column*s_byte+s_byte-1-i,row,15);
                tmp=tmp >> 1;
            }
        }
    }
    return;
}

/*****
 * converts from x,y coordinates to bit coordinates
 *****/

```

```

* x and y must both be in meters
*****

```

```

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;
    int j=ones,k=0;

    /* a is an integer which signifies which column array element in the
    * map has the column bit for the left bit.
    */
    a=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=floor((r_bit.col_bit)/s_byte);

/* if an entire integer (16 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;  
}  
}
```

APPENDIX B: ADAPT SIMULATION SOFTWARE CODE


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

```
FILE      : ZZ_CAN.C
```

```
DESCRIPTION  : CONTROLLER AREA NETWORK FUNCTIONS  
              FOR AUTONOMOUS VEHICLE
```

```
              This file contains code for CAN COMMUNICATIONS
```

```
-----  
by        : Ricardo Andujar
```

```
LAST UPDATE  : MARCH 22, 1993
```

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

```
#include "ZZ_CAN.H"
```

```
#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);
}

```

```

/* 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;
```

```

/*****
FILE      : ZZ_CONSL.C

DESCRIPTION  : CONSOLE MODULE FOR AUTONOMOUS VEHICLE

                This file contains code for KEYBOARD PROCESSING
-----

by      : Ricardo Andujar

LAST UPDATE   : MARCH 22, 1993
*****/

```

```

/*****
INCLUDE FILES FOR CONSOLE MODULE
*****/

```

```

#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#include <stdio.h>
#include <bios.h>          /* Microsoft specific */
#define FIRST_TIME 2
#define TRUE 1

```

```
extern int graph;
```

```
extern double ZZ_Circle;
```

```
double *WayX,*WayY;
```

```
static int *NewTarget;
```

```
int keypress,col=0,row=0;
```

```
/*CBUF *keybuf,*keybuf2;
*/
```

```
void ZZ_DrawCursor(void);
```

```

/*****
*
*   OBTAINS MEMORY ADDRESSES FOR WAYPOINT LOCATION VARIABLES   *
*
*****/

```

```

void ZZ_SupertoConsl(double *one,double *two, int *three)
{
    WayX = one;

```

```

WayY = two;
NewTarget = three;
}

```

```

/*****
*
*   KEYBOARD INTERRUPT
*
*****/
#define MASKPORT 1
void kbsig(void)
{
    int imask;          /* 8259 interrupt mask */

    imask = inportb(MASKPORT);    /* read current mask status */
    outportb(MASKPORT, 0xFF);    /* mask out all external interrupts */
    if (kbhit())                /* ASCII key available ? */
    {
        /* putcbuf(getch(),keybuf); */ /* get SCANCODE:ASCII */
        /* disable(); */ /* disable since bios enables interrupts */
        outportb(MASKPORT, imask); /* restore 8259 interrupt mask */
    }
    outportb(MASKPORT, imask);    /* restore 8259 mask */
}

```

```

/*****
*
*   RETURN KEY HANDLER
*
*****/
static void return_handler(void)
{
    /* col = 0;
    putcbuf("\0",keybuf2);
    setcolor(BLACK);
    outtextxy((col+1)*8,row*8,keybuf2->cb_front+1);
    setcolor(LIGHTGREEN);
    outtextxy(random(640),random(480),keybuf2->cb_front+1);
    resetcbuf(keybuf2);
    */
}

```

```

/*****
 *
 *   PRINT CHARACTER HANDLER
 *
 *****/
static void printchar_handler(void)
{
    /* int cc[2]={0,0};

    cc[0] = getchbuf(keybuf);
    if(++col>29)
    {
        unputcbuf(keybuf2);
        col=29;
    }
    else
    {
        setcolor(LIGHTGREEN);
        outtextxy(col*8,row*8,cc);
    }
    */
}

/*****
 *
 *   BACKSPACE HANDLER
 *
 *****/
static void backspace_handler(void)
{
    /* setcolor(BLACK);
    outtextxy(col*8,row*8,keybuf2->cb_rear);
    unputcbuf(keybuf2);
    if(--col<0)
        col=0;
    */
}

/*****
 *
 *   ESCAPE KEY HANDLER: Quits program
 *
 *****/
static void escape_handler()
{
    ZZ_EraseSuper();
    ZZ_EraseVision();
    ZZ_EraseNav();
    exit(1);
}

```

```

/*****
 *
 *   UP ARROW KEY HANDLER
 *
 *****/
void up_handler(void)
{
    ZZ_DrawCursor();
    *WayY+=.5;
    ZZ_DrawCursor();
}
/*****
 *
 *   DOWN ARROW KEY HANDLER
 *
 *****/
void down_handler(void)
{
    ZZ_DrawCursor();
    *WayY-=.5;
    ZZ_DrawCursor();
}
/*****
 *
 *   LEFT ARROW KEY HANDLER
 *
 *****/
void left_handler(void)
{
    ZZ_DrawCursor();
    *WayX-=.5;
    ZZ_DrawCursor();
}
/*****
 *
 *   RIGHT ARROW KEY HANDLER
 *
 *****/
void right_handler(void)
{
    ZZ_DrawCursor();
    *WayX+=.5;
    ZZ_DrawCursor();
}

/*****
 *
 *   DRAW CROSS HAIR CURSOR
 *
 *****/
void ZZ_DrawCursor(void)
{
    int wayx,wayy;

```



```

ZZ_Real2Screen(*WayX,*WayY,&wayx,&wayy);
setwritemode(1);
setcolor(LIGHTMAGENTA);
line(wayx-5,wayy,wayx+5,wayy);
line(wayx,wayy-5,wayx,wayy+5);
}
/*****
*
*   KEYPRESS MAIN HANDLER
*
*****/
void keypress_handler(void)
{
    static cc;
    if(kbhit())
    {
        cc = getch();
        switch(cc)
        {
            case 0:    switch(getch())
                        {
                            case 72: up_handler();
                                break;
                            case 80: down_handler();
                                break;
                            case 75: left_handler();
                                break;
                            case 77: right_handler();
                                break;
                        }
                    break;
            case '@': ZZ_Circle=1-ZZ_Circle;
                    break;
            case 27: escape_handler();
                    break;
            case 'M': print_map_handler();
                    break;
            case 'T': *NewTarget = FIRST_TIME;
                    break;
            /*      case 8:    backspace_handler();
                    break;
            case 13: return_handler();
                    break;
            default:  putchar(cc,keybuf2);
                    ungetcbuf(cc,keybuf);
                    printchar_handler();

                    break;
            */
        }
    }
}

```

```
/* consl.h */
```

```
void keypress_handler(void);  
extern void print_map_handler(void);  
void kbsig(void);
```

```
/******  
FILE      : ZZ_GRAPH.C
```

```
DESCRIPTION : SUPERVISOR MODULE FOR AUTONOMOUS VEHICLE
```

This file contains a graphics initialization and closing functions, conversion functions used to change between real and environment map coordinates, and functions to change from polar to cartesian coordinates and vice-versa.

```
-----  
by      : Ricardo Andujar
```

```
LAST UPDATE : MARCH 22, 1993
```

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

```
/******  
INCLUDE FILES FOR GRAPHIC FUNCTIONS FILE : SUPERVISOR MODULE  
*****/
```

```
#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"
```

```
/******  
*                                     *  
*   GRAPHIC INITIALIZATION FUNCTION   *  
*                                     *  
*****/
```

```
void ZZ_InitGraph(void)  
{  
    /* request auto detection */  
    int gdriver = DETECT, gmode, errorcode;  
  
    /* register a driver that was added into graphics.lib */  
    errorcode = registerbgidriver(EGAVGA_driver);  
  
    /* report any registration errors */  
    if (errorcode < 0)  
    {  
        printf("Graphics error: %s\n", grapherrormsg(errorcode));  
        exit(1); /* terminate with an error code */  
    }  
}
```

```

/* initialize graphics mode */
initgraph(&gdriver, &gmode, "");

/* read result
of initialization */
errorcode = graphresult();

if (errorcode != grOk) /* an error occurred */
{
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    exit(1); /* return with error code */
}
}

/*****
*
* GRAPHIC CLOSING FUNCTION
*
*****/
void ZZ_CloseGraph(void)
{
    closegraph();
}

/*****
*
* CHANGES FROM GRAPHIC TO REAL COORDINATES
* CARTESIAN COORDINATES
*
*****/
void ZZ_Screen2Real(int xc,int yc,double *x, double *y)
{
    *x = xc/SCREENCONVERT;
    *y = (BOTTOMLIMIT-yc)/SCREENCONVERT;
}

/*****
*
* CHANGES FROM REAL TO GRAPHIC COORDINATES
* CARTESIAN COORDINATES
*
*****/
void ZZ_Real2Screen(double x,double y,int *xc, int *yc)
{
    *xc = x*SCREENCONVERT;
    *yc = BOTTOMLIMIT-y*SCREENCONVERT;
}

```

```

/*****
 *
 *   CHANGES FROM CARTESIAN TO POLAR COORDINATES
 *
 *****/
void ZZ_Cart2Polar(double x,double y,double *ang,double *r)
{
    *ang = ZZ_Atan2(y,x);
    *r = ZZ_Range(y,x);
}

/*****
 *
 *   CHANGES FROM POLAR TO CARTESIAN COORDINATES
 *
 *****/
void ZZ_Polar2Cart(double ang,double r,double *x,double *y)
{
    *x = r*cos(ang);
    *y = r*sin(ang);
}

```

```
/*
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

```

```
-----

```

```
/*

```

```
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;
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(NAVIGATION,ZZ_NavServer);
ZZ_CAN_InstallServer(VISION,ZZ_VisionServer);

```

```
/*
***** Initializes Each Module *****

```

```
ZZ_InitSuper();
ZZ_InitVision();
ZZ_InitNav();
/* ZZ_CreateCircle(320,200);
ZZ_DrawCircle();
*/ ZZ_DrawCursor();

```

```
/*

```

```
***** Main Program Loop

```

```
***** -----

```

```
***** When finally impleteneted, 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.
*****/
    for(i=0;i<4;i++)
        ZZ_NavLoop();

/***** Vision Loop          *****/
    for(i=0;i<1;i++)
        ZZ_VisionLoop();

/***** Supervisor Loop      *****/
    for(i=0;i<1;i++)
        ZZ_SuperLoop();

/***** Handles Keyboard Presses *****/
    keypress_handler();
}
}

```

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

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

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

```
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"
```

```
extern  int  
graph;
```

```
staticdouble  
*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][40],ones; /* map[row][column] -- the actual map*/
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;

/*****
 *
 * 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=getmaxx() + 1;
    maxy=getmaxy() + 1;

    for(row=0;row<=479;++row)
        for(column=0;column<=39;++column)
            map[row][column]=ones;

    ZZ_DrawRoom();
    ZZ_DrawCar();
}

/*****
*
* 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;
        y = Yc[ii] + *Yconvert;

```

```

ZZ_Screen2Real((int)x,(int)y,&x,&y);
d=Range[ii];
r=0;
if (d >= max_dist)
    r=1;
phi=*Compass+NominalAngle[ii];

/* 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=6.28318+phi;
    else if (phi > 6.28318)
        phi=phi-6.28318;
    if (phi > 4.71 && phi < 4.72)
        phi=4.710;

w=d/cos(theta); /* w is length of triangle sides */

/* If heading angle is in the top half of the
* plane, define the triangle as following:
*/

if ((phi >= 0 && phi <= 1.57) || (phi > 4.71))
{
a=x+w*cos(1.57-theta-phi); /* (a,b) are coordinates of */
b=y+w*sin(1.57-theta-phi); /* right point on triangle */
if (b > -.01 && b < .01) /* round to zero if small */
    b=y;
p=x+w*sin(phi-theta); /* (p,q) are coordinates of */
q=y+w*cos(phi-theta); /* left point on triangle */
if(q > -0.01 && q < .01) /* round to zero if small*/
    q=0.0;
m1=(b-y)/(a-x); /* slope of line 1 (x,y) (a,b) */
if(m1 > -0.01 && m1 < 0.01) /* round to zero if small*/
    m1=0.0;
m2=(q-y)/(p-x); /* slope of line 2 (x,y) (p,q) */
if(m2 > -0.01 && m2 < 0.01) /* round to zero if small*/
    m2=0.0;
m3=(q-b)/(p-a); /* slope of line 3 (a,b) (p,q) */
if(m3 > -0.01 && m3 < 0.01) /* round to zero if small*/
    m3=0.0;
t1=y-x*m1; /* y-intercept for line 1 */
t2=y-x*m2; /* y-intercept for line 2 */
t3=q-p*m3; /* y-intercept for line 3 */
}

/* If heading angle is in the bottom half of the
* plane, define the triangle as following:
*/

```

```

else if (phi > 1.57 && phi <= 4.71)
{
a=x+w*sin(phi-theta);
b=y+w*cos(phi-theta);
if (b < .01 && b > -.01)
    b=y;
p=x+w*cos(1.57-theta-phi);
q=y+w*sin(1.57-theta-phi);
if(q < .01 && q > -.01)
    q=0.0;
m1=(b-y)/(a-x);
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;
m3=(q-b)/(p-a);
if(m3 > -0.01 && m3 < 0.01)
    m3=0.0;
t1=y-x*m1;
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 ((phi >= 0 && phi <= 1.57-theta) || (phi > 4.71+theta))
{

```

```

/* 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 present. 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 position
 * compared 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 ((phi > 1.57+theta) && (phi <= 4.71-theta))
    {

        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 < 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);
    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_row(l_bit,r_bit,t,r);
    tmp_bit=xy_to_bit(tmp_x,tmp_y-resolution);
}

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 of code clears triangles which
* are split by the horizontal plane.
*/

```

    else if ((phi > 1.57-theta && phi < 1.57+theta) ||
            (phi > 4.71-theta && phi < 4.71+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 (phi < 3.14159)
    {

```

```

        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 orientation of the triangle and
 * specifies the line to which the next
 * row originates.
 */

```

```

    if (phi > 1.57)
    {
        if (m1 == 0)
            tmp_x=tmp_x+resolution;
        else
            tmp_x=(tmp_y-t1)/m1;
    }
    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 to next row */

```

```

    tmp_y=y-resolution;
    if (phi > 1.57)
    {
        if (m2 == 0)
            tmp_x=tmp_x+resolution;
        else

```



```

        tmp_x=(tmp_y-t2)/m2;
    }
else
    {
    if (m1 == 0)
        tmp_x=tmp_x+resolution;
    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 > 1.57)
        {
            if (m2 == 0)
                tmp_x=tmp_x+resolution;
            else
                tmp_x=(tmp_y-t2)/m2;
        }
        else
        {
            if (m1 == 0)
                tmp_x=tmp_x+resolution;
            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 (phi > 3.14159)
    {
        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 < 4.72)
{
    if (m2 == 0)
        tmp_x=tmp_x+resolution;
    else
        tmp_x=(tmp_y-t2)/m2;
}
else
{
    if (m1 == 0)
        tmp_x=tmp_x+resolution;
    else
        tmp_x=(tmp_y-t1)/m1;
}
tmp_bit=xy_to_bit(tmp_x,tmp_y);
}
tmp_y=y-resolution;
if (phi > 4.71)
{
    if (m2 == 0)
        tmp_x=tmp_x+resolution;
    else
        tmp_x=(tmp_y-t2)/m2;
}
else
{
    if (m1 == 0)
        tmp_x=tmp_x+resolution;
    else
        tmp_x=(tmp_y-t1)/m1;
}

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 > 4.71)
    {
        if (m2 == 0)
            tmp_x=tmp_x+resolution;
        else
            tmp_x=(tmp_y-t2)/m2;
    }
    else
    {

```



```

/* 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;
    }

}

/*****
 *      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);
}

/*****
 * print map handler
 *****/

void print_map_handler(void)
{
    int i,mask=1,row,column;
    unsigned int tmp;

//    cleardevice();
    for (row=0;row<maxy;++row)
        {
            for (column=0;column<maxx;++column)
                {
                    tmp=map[row][column];
                    for (i=0;i<s_byte;++i)

```

```

void clear_row(bit l_bit,bit r_bit,int t,int r)
{
    int a,b,c,i;
    int j=ones,k=0;

/* 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;
}

}

/*****
 *      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);
}

/*****
 * print map handler *
 *****/

void print_map_handler(void)
{
    int i,mask=1,row,column;
    unsigned int tmp;

//    cleardevice();
    for (row=0;row<maxy;++row)
        {
            for (column=0;column<maxx;++column)
                {
                    tmp=map[row][column];
                    for (i=0;i<s_byte;++i)

```

```

        {
        if(tmp & mask)
            putpixel(column*s_byte+s_byte-1-i,row,2);
        else
            putpixel(column*s_byte+s_byte-1-i,row,15);
        tmp=tmp >> 1;
        }
    }
}
getch();
cleardevice();
ZZ_DrawRoom();
ZZ_DrawCar();

}

```

```

/*****
*
*   DRAW ROBOT EDGES.
*
*****/

```

```

void ZZ_DrawCar(void)
{
    int i;
    setwritemode(1);
    setcolor(LIGHTMAGENTA);
    moveto(*Xconvert+CarXc[0],*Yconvert+CarYc[0]);
    for(i=1;i<NUM_EDGES;i++)
    {
        lineto(*Xconvert+CarXc[i],*Yconvert+CarYc[i]);
    }
}

```

```

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);
}

```

```

/*****
*
*   DRAW ROOM FUNCTION. ONLY FOR SIMULATION !!
*
*****/

```

```

*
*****
void ZZ_DrawRoom(void)
{
    int i,j;
    setwritemode(0);
    setcolor(WHITE);
    setfillstyle(SOLID_FILL,WHITE);
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    bar(LEFTLIMIT,TOPLIMIT,RIGHTLIMIT,BOTTOMLIMIT);
    setfillstyle(SOLID_FILL,LIGHTGRAY);
    bar(LEFTLIMIT+10,TOPLIMIT+10,RIGHTLIMIT-10,BOTTOMLIMIT-10);
    setfillstyle(SOLID_FILL,WHITE);

/*   {   int x,y,bx=55,by=55;
    x = 100; y = 100;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 100; y = 300;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 400; y = 100;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 400; y = 300;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 100; by = 20; y = (100+300+bx-by)/2.0; bx = 300+bx;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = (100+400+55-20)/2.; y = 100; by = 55; bx = 20; by = 200+by;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    }

    THREE
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 100; y = 230;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 225; y = 70;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 225; y = 170;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 225; y = 270;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 350; y = 110;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 350; y = 230;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 475; y = 70;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 475; y = 170;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    x = 475; y = 270;
    bar(LEFTLIMIT+x,TOPLIMIT+y,LEFTLIMIT+x+bx,TOPLIMIT+y+by);
    }

    NUMBER FOUR
*/   bar(LEFTLIMIT+45+30,TOPLIMIT+45+30,LEFTLIMIT+100+30,TOPLIMIT+100+30);
    bar(LEFTLIMIT+45+40,TOPLIMIT+45+190,LEFTLIMIT+100+40,TOPLIMIT+100+190);

```



```
bar(LEFTLIMIT+45+100,TOPLIMIT+45+130,LEFTLIMIT+100+100,TOPLIMIT+100+130);  
bar(LEFTLIMIT+45+100,TOPLIMIT+45+260,LEFTLIMIT+100+100,TOPLIMIT+100+260);
```

```
bar(LEFTLIMIT+30+100,BOTTOMLIMIT-100,LEFTLIMIT+100+100,BOTTOMLIMIT-10);
```

```
bar(LEFTLIMIT+250,TOPLIMIT+60,LEFTLIMIT+270,BOTTOMLIMIT-60);  
bar(LEFTLIMIT+320,TOPLIMIT,LEFTLIMIT+340,BOTTOMLIMIT-250);  
bar(LEFTLIMIT+320,TOPLIMIT+250,LEFTLIMIT+340,BOTTOMLIMIT);  
bar(LEFTLIMIT+270,BOTTOMLIMIT-270,LEFTLIMIT+320,BOTTOMLIMIT-250);
```

```
bar(LEFTLIMIT+390,BOTTOMLIMIT-300,LEFTLIMIT+450,BOTTOMLIMIT-180);
```

```
bar(LEFTLIMIT+390,BOTTOMLIMIT-160,LEFTLIMIT+410,BOTTOMLIMIT-60);  
bar(LEFTLIMIT+390,BOTTOMLIMIT-80,LEFTLIMIT+510,BOTTOMLIMIT-60);
```

```
bar(LEFTLIMIT+340,TOPLIMIT+80,LEFTLIMIT+530,TOPLIMIT+100);  
bar(LEFTLIMIT+510,TOPLIMIT+100,LEFTLIMIT+530,TOPLIMIT+280);
```

```
bar(LEFTLIMIT+550,TOPLIMIT+330,LEFTLIMIT+570,TOPLIMIT+400);
```

```
}
```

```

/* 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);

```

```

/*****
FILE      : ZZ_MISC.C

DESCRIPTION  : MISCELLANEOUS FUNCTION'S FILE FOR AUTONOMOUS VEHICLE

```

This file contains code for assorted functions
used through out the program.

```

by      : Ricardo Andujar

```

```

LAST UPDATE  : MARCH 22, 1993

```

```

/*****
INCLUDE FILES FOR miscellaneous file
*****/
#include<math.h>

```

```

/*****
*
*   SWAPS TWO NUMBERS
*
*****/

```

```

void ZZ_Swap(double *one,double *two)
{
    double temp;

    temp = *one;
    *one = *two;
    *two = temp;
}

```

```

/*****
*
*   LIMIT'S ANGLE TO -180<pi<180 (DEGREES)
*
*****/

```

```

void ZZ_LimitAngle(double *Angle)
{
    if(*Angle>M_PI)
        *Angle -= 2*M_PI;
    else
        if(*Angle<-M_PI)
            *Angle += 2*M_PI;
}

```

```

/*****
 *
 * ERROR PROOF TANGENT FUNCTION : Avoids divide by zero      *
 *
 *****/
double  ZZ_Atanh2(double y, double x)
{
    if(x!=0.0)
        return(atan2(y,x));
    else if(y>0)
        return(M_PI_2);
    else
        return(-M_PI_2);
}

/*****
 *
 * RETURNS DISTANCE FROM ZERO COORDINATE                      *
 *
 *****/
double  ZZ_Range(double x,double y)
{
    return(sqrt(x*x+y*y));
}

```

```
/* 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);
```

```

/*****
FILE      : ZZ_NAV.C

DESCRIPTION  : MAIN FILE FOR PROPULSION MODULE OF AUTONOMOUS VEHICLE
-----

```

```

by      : Ricardo Andujar

```

```

LAST UPDATE   : MARCH 22, 1993

```

```

*****/

```

```

/*****

```

```

INCLUDE FILES FOR MAN

```

```

*****/

```

```

#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"

```

```

/***** GLOBAL DEFINITIONS *****/

```

```

extern int graph;

```

```

/***** PRIVATE DEFINITIONS *****/

```

```

static void ZZ_UpdateSensorMeas(void);
static void ZZ_CarSim(void);
static void ZZ_UpdateActuators(void);

```

```

#define ON 1

```

```

static unsigned short Brake=ON;

```

```

static double  TimeStep=0.050,  /**** SAMPLE PERIOD (seconds) ****/

```

```

    RoadSpeed=0,
    Compass=0.0,
    Xposition=1.0,
    Yposition=4.0,
    RoadSpeed2=0,
    Xposition2=1.0,
    Yposition2=4.0,
    Compass2=0,
    CruiseSpeed=0,
    MotorTorque=0,
    k1=0,

```

```
k2=0,  
k3=0,  
k4=0,
```

```
*****  
***** PI-CONTROL PARAMETERS FOR DC-Motor  
***** This is only for simulation purposes  
***** Actual Control of vehicle may change  
*****/
```

```
MKP=4,  
MKI=0.004,  
SpeedError=0,  
SpeedSum=0,
```

```
ASteeringAngle=0,  
ASteeringAngle2=0,  
DSteeringAngle=0,  
CarXc[NUM_EDGES],  
CarYc[NUM_EDGES],  
CarR[NUM_EDGES],  
CarAng[NUM_EDGES];
```

```
*****  
* * * * *  
* Information needed for Vision Simulation. *  
* This function is only needed for simulation purposes *  
* * * * *  
*****/
```

```
void ZZ_NavToVision(long *one,long *two,long *three)  
{  
    *one = &Xposition2;  
    *two = &Yposition2;  
    *three = &Compass2;  
}
```

```
*****  
* * * * *  
* PROPULSION LOOP *  
* * * * *  
*****/
```

```
void ZZ_NavLoop(void)  
{  
    ZZ_UpdateSensorMeas();  
    ZZ_UpdateActuators();  
}
```

```
*****  
* * * * *
```

```

*   UPDATES ALL ACTUATORS, POSITION AND BEARING           *
*   INFORMATION                                         *
*
*****
void ZZ_UpdateSensorMeas(void)
{
    /******
    ***** 'ZZ_CarSim' is for Simulation Purposes Only.
    ***** Function used TO OBTAIN SENSOR INFO Goes Here
    *****/
    ZZ_CarSim();
}

/*****
*
*   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;
    }
}

/*****
*
*   ALL THE CONTROLS FOR THE ACTUATORS GO HERE         *
*
*****

```



```

void ZZ_UpdateActuators(void)
{
/*****
**** 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.
**** */
    SpeedError = CruiseSpeed-RoadSpeed;
    SpeedSum += SpeedError;
    if (SpeedSum>1)
        SpeedSum= 1;
    else
    if (SpeedSum<-1)
        SpeedSum=-1;
    MotorTorque = MKP*SpeedError +MKI*SpeedSum;
    ASteeringAngle = DSteeringAngle;
}

```

```

/*****
*
* CAR SIMULATION ONLY
*
**** */

```

```

void ZZ_CarSim(void)
{
/*****
**** 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.
**** */
    void ZZ_CheckCrash(void);

    Compass2=Compass;
    Xposition2 = Xposition;
    Yposition2 = Yposition;
    RoadSpeed2 = RoadSpeed;
    ASteeringAngle2 = ASteeringAngle;

    Compass+=TimeStep*RoadSpeed*sin(ASteeringAngle)/AxleToAxleLength;
    ZZ_LimitAngle(&Compass);
    Xposition+=TimeStep*RoadSpeed*sin(Compass+ASteeringAngle);
    Yposition+=TimeStep*RoadSpeed*cos(Compass+ASteeringAngle);

    if(Brake)

```

```

    {
        k1 = (MotorTorque-MotorViscosity*RoadSpeed)/MotorInertia;
        k2 = (MotorTorque-MotorViscosity*(RoadSpeed+.5*k1))/MotorInertia;
        k3 = (MotorTorque-MotorViscosity*(RoadSpeed+.5*k2))/MotorInertia;
        k4 = (MotorTorque-MotorViscosity*(RoadSpeed+k3))/MotorInertia;
    }
    else
    {
        k1 = (MotorTorque-Brake*RoadSpeed)/MotorInertia;
        k2 = (MotorTorque-Brake*(RoadSpeed+.5*k1))/MotorInertia;
        k3 = (MotorTorque-Brake*(RoadSpeed+.5*k2))/MotorInertia;
        k4 = (MotorTorque-Brake*(RoadSpeed+k3))/MotorInertia;
    }
    RoadSpeed+=TimeStep/6.0*(k1+k4+2.0*(k2+k3));
/* ZZ_CheckCrash();
*/}

/*****
*
* CALL THIS FUNCTION WHEN INITIALIZING PROPULSION MODULE *
*
*****/
void ZZ_InitNav(void)
{
    int i;
    for(i=0;i<NUM_EDGES;i++)
        ZZ_Cart2Polar(CarEdgeX[i],CarEdgeY[i],&CarAng[i],&CarR[i]);
}

/*****
*
* CALL THIS FUNCTION WHEN QUITTING PROGRAM *
*
*****/
void ZZ_EraseNav(void)
{
}

/*****
*
* SIMULATION TO CHECK FOR CRASHES *
*
*****/
void ZZ_CheckCrash(void)
{
    static int Crash;
    int i,color,Xconvert,Yconvert;

    ZZ_Real2Screen(Xposition,Yposition,&Xconvert,&Yconvert);
    for(i=0;i<NUM_EDGES;i++)

```

```
{
    ZZ_Polar2Cart(CarAng[i]+Compass,CarR[i]*SCREENCONVERT,
                  &CarXc[i],&CarYc[i]);

    color = getpixel(Xconvert+CarXc[i],Yconvert+CarYc[i]);
    if ((--Crash)<0 &&(color == WHITE || color == BLUE
                    || color ==LIGHTRED || color == GREEN))
    {
        RoadSpeed = -RoadSpeed*1;
        Crash = 0;
        return;
    }
}
```

```
/* ZZ_nav.h */
```

```
void ZZ_NavLoop(void);
```

```
void ZZ_NavServer(byte SourceAddress,int *DataContent,  
                  double *Data, byte *DataNum);
```

```

/*****
FILE      : ZZ_OBJCT.C

DESCRIPTION  : MOVING OBJECT SIMULATION FILE FOR AUTONOMOUS VEHICLE

```

This file contains code to display moving objects.

by : Ricardo Andujar

LAST UPDATE : MARCH 22, 1993

```

/*****

```

```

/*****
INCLUDE FILES FOR MOVING OBJECT FILE :SUPERVISOR MODULE
*****

```

```

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>

```

```

extern int graph;

```

```

int  CircleX=320,
     CircleY=250,
     Flag=0;

```

```

void far *bitcir;

```

```

unsigned size;

```

```

/*****

```

```

*
*   DRAWS CIRCLE ON SCREEN
*
*****

```

```

void ZZ_DrawCircle(void)
{
    getimage(CircleX-25,CircleY-25,CircleX+25,CircleY+25,bitcir);
    setwritemode(1);
    setcolor(BLACK);
    setfillstyle(SOLID_FILL,LIGHTRED);
    fillellipse(CircleX,CircleY,25,25);
}

```

```

/*****
 *
 *   ERASES CIRCLE FROM SCREEN
 *
 *****/

```

```

void ZZ_EraseCircle(void)
{
    while(Flag);
    putimage(CircleX-25,CircleY-25,bitcir,0);
}

```

```

/*****
 *
 *   ALLOCATES MEMORY FOR CIRCLE ON SCREEN
 *
 *****/

```

```

void ZZ_CreateCircle(int x,int y)
{
    CircleX = x;
    CircleY = y;
    size = imagesize(0, 0, 50, 50); /* get byte size of image */
    bitcir = farmalloc(size);
}

```

```

/*****
 *
 *   DEALLOCATES MEMORY FOR CIRCLE ON SCREEN
 *
 *****/

```

```

void ZZ_DestroyCircle(void)
{
    farfree(bitcir);
}

```

```
/* ZZ_objct.h */
```

```
void ZZ_CreateCircle(int x,int y);  
void ZZ_DestroyCircle(void);  
void ZZ_EraseCircle(void);  
void ZZ_DrawCircle(void);
```

```

/* ZZ_planr.c */

#include<graphics.h>
#include<alloc.h>

#include"ZZ_QUEUE.H"

#define LEAVE_LIMIT 15
#define box 15
#define LMAX 400
#define LMAX2 1000
#define ON 1

#define precision 3 /*SEARCH RESOLUTION */
#define precision2 15 /* FINAL TARGET DISTANCE ALLOWANCE */

typedef struct
{
    int    x,
           y,
           north,
           northeast,
           east,
           southeast,
           south,
           southwest,
           west,
           northwest;
} ZZ_ExtendedPoint;

int  j,x=30,
     y=385,
     tx=560,
     ty=236,
     numcw=0,numccw=0,result,Lx,Ly,Hx,Hy;

extern int Adapt;

ZZ_ExtendedPoint  Cw,Ccw;
ZZ_Point          PCw,PCcw,Ptarget;
QUEUE             QMain={0,0},QCw={0,0},Qccw={0,0};

void ZZ_FollowWallClockwise(ZZ_ExtendedPoint *p);
void ZZ_FollowWallCounterClockwise(ZZ_ExtendedPoint *p);

int ZZ_Planner(int x, int y,int tx,int ty,QUEUE *PATH)
{
    int stat=0,Cwu=0,Ccwu=0,limitCw=0,limitccw=0;

```



```

****
****          ****
****    GO AROUND OBSTACLE COUNTERCLOCKWISE AND    ****
****    CLOCKWISE UNTIL REACHING LEAVE POINT.      ****
****          ****
*****/
if(result == 0 || result == 2)
while(1)
{
    ZZ_FollowWallClockwise(&Cw);
    ZZ_FollowWallCounterClockwise(&Ccw);
    PCw.x = Cw.x;
    PCw.y = Cw.y;
    PCcw.x = Ccw.x;
    PCcw.y = Ccw.y;
    enqueue(&QCw,&PCw);
    enqueue(&QCcw,&PCcw);
    if(result == 2)
        if(++limitCw > LMAX)
        {
            if(endtotarget(&QCw,&PTarget) <
                endtotarget(&QCcw,&PTarget))
            {
                while(!isempty(&QCw))
                {
                    dequeue(&QCw,&PCw);
                    enqueue(&QMain,&PCw);
                }
            }
            else
            {
                while(!isempty(&QCcw))
                {
                    dequeue(&QCcw,&PCcw);
                    enqueue(&QMain,&PCcw);
                }
            }
            QCw.front = QCw.rear = NULL;
            resetqueue(&QCw);
            QCcw.front = QCcw.rear = NULL;
            resetqueue(&QCcw);
            Cwu = 1;
            break;
        }
    if(result == 0)
    {
        if(++limitccw > LMAX2)
        {
            if(endtotarget(&QCw,&PTarget) <
                endtotarget(&QCcw,&PTarget))
            {
                while(!isempty(&QCw))
                {

```

```

        dequeue(&QCw,&PCw);
        enqueue(&QMain,&PCw);
    }
}
else
{
    while(!isempty(&QCcw))
    {
        dequeue(&QCcw,&PCcw);
        enqueue(&QMain,&PCcw);
    }
}
QCw.front = QCw.rear = NULL;
resetqueue(&QCw);
QCcw.front = QCcw.rear = NULL;
resetqueue(&QCcw);
Cwu = 1;
break;
}

if(abs(Lx-Cw.x)<LEAVE_LIMIT && abs(Ly-Cw.y)<LEAVE_LIMIT)
{
    while(!isempty(&QCw))
    {
        dequeue(&QCw,&PCw);
        enqueue(&QMain,&PCw);
    }
    QCw.front = QCw.rear = NULL;
    resetqueue(&QCw);
    QCcw.front = QCcw.rear = NULL;
    resetqueue(&QCcw);
    break;
}
if(abs(Lx-Ccw.x)<LEAVE_LIMIT && abs(Ly-Ccw.y)<LEAVE_LIMIT)
{
    while(!isempty(&QCcw))
    {
        dequeue(&QCcw,&PCcw);
        enqueue(&QMain,&PCcw);
    }
    QCw.front = QCw.rear = NULL;
    resetqueue(&QCw);
    QCcw.front = QCcw.rear = NULL;
    resetqueue(&QCcw);
    break;
}
}
} /** END OF WHILE LOOP **/

/**
*** Either reached target or stop after specified iterations
***/

if(Cwu)

```

```

        break;
    }

    /**
    *** If condition is true than target was
    *** not reached
    ***/
    if(Cwu)
    {
        setcolor(YELLOW);
        qplot(&QMain);
        stat = 1;
    }
    /**
    *** Optimize path
    ***/
    if(!stat)
    {
        while(optimize(&QMain));
        setcolor(LIGHTMAGENTA);
        qplot(&QMain);
        gotoxy(1,1);printf("T");
        getch();
        qplot(&QMain);
        Adapt=1;
    }
    /**
    *** Plot optimized path.
    ***/
    PATH->front = QMain.front;
    PATH->rear = QMain.rear;
    QMain.front = QMain.rear = NULL;
    return(stat);
}

```

```

void ZZ_FollowWallClockwise(ZZ_ExtendedPoint *p)
{
    /**
    if(!p->north)
    while(1)
    {
        p->x -= precision;
        p->y -= precision;
        for(j=0;j<=box;j+=4)
        if(getbit(p->x-j,p->y-j) == ON ||
            getbit(p->x-j,p->y) == ON ||

```

```

        getbit(p->x,p->y-j) == ON )
    {
        p->north = 1;
        p->northwest = 0;
        p->x += precision;
        p->y += precision;
        return;
    }
    p->y -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x,p->y-j) == ON ||
        getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x-j,p->y-j) == ON )
        p->y += precision;
    else
    {
        p->north = 1;
        p->northeast = 0;
        return;
    }
}

```

/******

```

if(!p->northwest)
while(1)
{
    p->x-=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y) == ON ||
        getbit(p->x-j,p->y+j) == ON ||
        getbit(p->x-j,p->y-j) == ON )
        {
            p->x+=precision;
            p->northwest = 1;
            p->west = 0;
            return;
        }
    p->x -= precision;
    p->y -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y-j) == ON ||
        getbit(p->x-j,p->y) == ON ||
        getbit(p->x,p->y-j) == ON )
        {
            p->x += precision;
            p->y += precision;
        }
    else
    {
        p->northwest = 1;
        p->north = 0;
        return;
    }
}
}

```

```

/*****/
if(!p->west)
while(1)
{
    p->y+=precision;
    p->x-=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y+j) == ON ||
        getbit(p->x-j,p->y) == ON ||
        getbit(p->x,p->y+j) == ON )
        {
            p->west = 1;
            p->southwest = 0;
            p->y -= precision;
            p->x += precision;
            return;
        }
    p->x -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y) == ON ||
        getbit(p->x-j,p->y+j) == ON ||
        getbit(p->x-j,p->y-j) == ON )
        p->x += precision;
    else
    {
        p->west = 1;
        p->northwest = 0;
        return;
    }
}
/*****/
if(!p->southwest)
while(1)
{
    p->y += precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x,p->y+j) == ON ||
        getbit(p->x+j,p->y+j) == ON ||
        getbit(p->x-j,p->y+j) == ON )
        {
            p->southwest = 1;
            p->south = 0;
            p->y -= precision;
            return;
        }
    p->x -= precision;
    p->y += precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y+j) == ON ||
        getbit(p->x-j,p->y) == ON ||
        getbit(p->x,p->y+j) == ON )
        {

```

```

        p->x += precision;
        p->y -= precision;
    }
    else
    {
        p->southwest = 1;
        p->west = 0;
        return;
    }
}

```

/***/

```

if(!p->south)
while(1)
{
    p->x+=precision;
    p->y+=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y+j) == ON ||
        getbit(p->x+j,p->y) == ON ||
        getbit(p->x,p->y+j) == ON )
    {
        p->south = 1;
        p->southeast = 0;
        p->x -= precision;
        p->y -= precision;
        return;
    }
    p->y+=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x,p->y+j) == ON ||
        getbit(p->x+j,p->y+j) == ON ||
        getbit(p->x-j,p->y+j) == ON )
        p->y-=precision;
    else
    {
        p->south = 1;
        p->southwest = 0;
        return;
    }
}

```

/***/

```

if(!p->southeast)
while(1)
{
    p->x += precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y) == ON ||
        getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x+j,p->y+j) == ON )
    {
        p->x -= precision;
        p->southeast = 1;
    }
}

```

```

        p->east = 0;
        return;
    }
    p->x += precision;
    p->y += precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y+j) == ON ||
        getbit(p->x+j,p->y) == ON ||
        getbit(p->x,p->y+j) == ON )
    {
        p->x -= precision;
        p->y -= precision;
    }
    else
    {
        p->southeast = 1;
        p->south = 0;
        return;
    }
}
/*****/
if(!p->east)
while(1)
{
    p->x += precision;
    p->y -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x,p->y-j) == ON ||
        getbit(p->x+j,p->y) == ON )
    {
        p->x -= precision;
        p->y += precision;
        p->east = 1;
        p->northeast = 0;
        return;
    }
    p->x+=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y) == ON ||
        getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x+j,p->y+j) == ON )
        p->x -= precision;
    else
    {
        p->east = 1;
        p->southeast = 0;
        return;
    }
}
/*****/
if(!p->northeast)
while(1)

```



```

    {
        p->y -= precision;
        for(j=0;j<=box;j+=4)
            if(getbit(p->x,p->y-j) == ON ||
                getbit(p->x+j,p->y-j) == ON ||
                getbit(p->x-j,p->y-j) == ON )
                {
                    p->y += precision;
                    p->northeast = 1;
                    p->north = 0;
                    return;
                }
        p->x += precision;
        p->y -= precision;
        for(j=0;j<=box;j+=4)
            if(getbit(p->x+j,p->y-j) == ON ||
                getbit(p->x+j,p->y) == ON ||
                getbit(p->x,p->y-j) == ON )
                {
                    p->x -= precision;
                    p->y += precision;
                }
        else
            {
                p->northeast = 1;
                p->east = 0;
                return;
            }
    }
}
}

```

```

void ZZ_FollowWallCounterClockwise(ZZ_ExtendedPoint *p)
{
    /******
    if(!p->east)
        while(1)
        {
            p->y+=precision;
            p->x+=precision;
            for(j=0;j<=box;j+=4)
                if(getbit(p->x+j,p->y+j) == ON ||
                    getbit(p->x,p->y+j) == ON ||
                    getbit(p->x+j,p->y) == ON )

```

```

    {
        p->east = 1;
        p->southeast = 0;
        p->x -= precision;
        p->y -= precision;
        return;
    }
    p->x+=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y) == ON ||
        getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x+j,p->y+j) == ON )
        p->x -= precision;
    else
    {
        p->east = 1;
        p->northeast = 0;
        return;
    }
}

```

/******

```

if(!p->southeast)
while(1)
{
    p->y+=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x,p->y+j) == ON ||
        getbit(p->x+j,p->y+j) == ON ||
        getbit(p->x-j,p->y+j) == ON )
    {
        p->southeast = 1;
        p->south = 0;
        p->y -= precision;
        return;
    }
    p->x+=precision;
    p->y+=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y+j) == ON ||
        getbit(p->x,p->y+j) == ON ||
        getbit(p->x+j,p->y) == ON )
    {
        p->x -= precision;
        p->y -= precision;
    }
    else
    {
        p->southeast = 1;
        p->east = 0;
        return;
    }
}
}

```

```

/*****/
if(!p->south)
while(1)
{
    p->x -= precision;
    p->y += precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y+j) == ON ||
        getbit(p->x,p->y+j) == ON ||
        getbit(p->x-j,p->y) == ON )
        {
            p->south = 1;
            p->southwest = 0;
            p->x += precision;
            p->y -= precision;
            return;
        }
    p->y+=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x,p->y+j) == ON ||
        getbit(p->x+j,p->y+j) == ON ||
        getbit(p->x-j,p->y+j) == ON )
        p->y-=precision;
    else
    {
        p->south = 1;
        p->southeast = 0;
        return;
    }
}

```

```

/*****/
if(!p->southwest)
while(1)
{
    p->x -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y) == ON ||
        getbit(p->x-j,p->y+j) == ON ||
        getbit(p->x-j,p->y-j) == ON )
        {
            p->southwest = 1;
            p->west = 0;
            p->x += precision;
            return;
        }
    p->x -= precision;
    p->y += precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y+j) == ON ||
        getbit(p->x,p->y+j) == ON ||
        getbit(p->x-j,p->y) == ON )

```

```

    {
        p->x += precision;
        p->y -= precision;
    }
    else
    {
        p->southwest = 1;
        p->south = 0;
        return;
    }
}

```

/***/

```

if(!p->west)
while(1)
{
    p->x -= precision;
    p->y -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y-j) == ON ||
        getbit(p->x-j,p->y) == ON ||
        getbit(p->x,p->y-j) == ON )
    {
        p->west = 1;
        p->northwest = 0;
        p->x += precision;
        p->y += precision;
        return;
    }
    p->x -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x-j,p->y) == ON ||
        getbit(p->x-j,p->y+j) == ON ||
        getbit(p->x-j,p->y-j) == ON )
        p->x += precision;
    else
    {
        p->west = 1;
        p->southwest = 0;
        return;
    }
}

```

/***/

```

if(!p->northwest)
while(1)
{
    p->y -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x,p->y-j) == ON ||
        getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x-j,p->y-j) == ON )

```

```

{
    p->y += precision;
    p->northwest = 1;
    p->north = 0;
    return;
}
p->x -= precision;
p->y -= precision;
for(j=0;j<=box;j+=4)
if(getbit(p->x-j,p->y-j) == ON ||
    getbit(p->x,p->y-j) == ON ||
    getbit(p->x-j,p->y) == ON )
{
    p->x += precision;
    p->y += precision;
}
else
{
    p->northwest = 1;
    p->west = 0;
    return;
}
}

```

/******

```

if(!p->north)
while(1)
{
    p->x+=precision;
    p->y-=precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x+j,p->y) == ON ||
        getbit(p->x,p->y-j) == ON )
    {
        p->north = 1;
        p->northeast = 0;
        p->x -= precision;
        p->y += precision;
        return;
    }
    p->y -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x,p->y-j) == ON ||
        getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x-j,p->y-j) == ON )
        p->y += precision;
    else
    {
        p->north = 1;
        p->northwest = 0;
        return;
    }
}

```

```

    }
}

/*****/
if(!p->northeast)
while(1)
{
    p->x += precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y) == ON ||
        getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x+j,p->y+j) == ON )
    {
        p->x -= precision;
        p->east = 0;
        p->northeast = 1;
        return;
    }
    p->x += precision;
    p->y -= precision;
    for(j=0;j<=box;j+=4)
    if(getbit(p->x+j,p->y-j) == ON ||
        getbit(p->x,p->y-j) == ON ||
        getbit(p->x+j,p->y) == ON )
    {
        p->x -= precision;
        p->y += precision;
    }
    else
    {
        p->northeast = 1;
        p->north = 0;
        return;
    }
}
}

```

```

/*****
FILE      : ZZ_QUEUE.C

DESCRIPTION : CONTAINS ALL QUEUE RELATED FUNCTIONS USED BY
            THE PATH PLANNER.

-----

by       : Ricardo Andujar

LAST UPDATE : MARCH 22, 1993
*****/

```

```

/*****
INCLUDE FILES FOR QUEUE
*****/

```

```

#include<stdlib.h>
#include<alloc.h>
#include<conio.h>
#include<graphics.h>
#include<stdio.h>
#include<math.h>
#include "zz_queue.h"

```

```

/*****
*****      This variable can be modified to change
*****      path search precision
*****/

```

```

#define precision2 1

```

```

#define NULL 0
#define ON 1
extern int getbit(int,int);

```

```

/*****
*
* RETURNS A 1 IF QUEUE IS EMPTY, OTHERWISE RETURNS 0.
*
*****/

```

```

int isempty(QUEUE *q)
{
    return(q->front == NULL);
}

```

```

/*****
*
* RETURNS THE FIRST ELEMENT IN THE QUEUE WITHOUT UNQUEUEING
*
*****/

```

```

DATA vfront(QUEUE *q)
{

```

```

return (q->front -> d);
}

```

```

/*****
*
* RETURNS 1 IF FINDS VALUE EQUAL TO SESARCH ARGUMENT *
*
*****/

```

```

int qsearch(QUEUE *q, DATA *x)
{
LINK temp = q -> rear;
if(temp->previous == q->front ||
temp->previous->previous == q->front)
return(0);
temp = temp->previous->previous;
while(temp -> previous != NULL)
{
if(temp->d.x == x->x && temp->d.y == x->y)
return(1);
temp = temp -> previous;
}
return(0);
}

```

```

/*****
*
* PLOTS ENTIRE PATH GENERATED *
*
*****/

```

```

void qplot(QUEUE *q)
{
LINK temp = q -> front;

gotoxy(1,1);printf("p");
moveto(temp->d.x,temp->d.y);
while(temp != NULL)
{
lineto(temp->d.x,temp->d.y);
temp = temp -> next;
}
}

```

```

/*****
*
* EMPTIES QUEUE *
*
*****/

```

```

void resetqueue(QUEUE *q)
{

```



```

LINK temp = q -> front;
gotoxy(1,1);printf("r");
while(!isempty(q))
{
    q -> front = temp -> next;
    q -> front -> previous = NULL;
    farfree(temp);
    temp = q-> front;
}
}

```

```

/*****
*
* RETURNS NUMBER OF ELEMENTS IN QUEUE
*
*****/

```

```

int numqueue(QUEUE *q)
{
    int num=0;
    LINK temp;
    temp = q->front;
    while(temp != NULL)
    {
        num++;
        temp = temp->next;
    }
    return(num);
}

```

```

/*****
*
* RETURNS THE SUM OF THE DISTANCES BETWEEN ALL
* ADJACENT WAYPOINTS
*
*****/

```

```

int sumqueue(QUEUE *q)
{
    int num=0;
    LINK temp;
    temp = q->front;
    while(temp->next != NULL)
    {
        num += sqrt(pow(temp->d.x - temp->next->d.x,2.0L)+
                    pow(temp->d.y - temp->next->d.y,2.0L));    temp = temp->next;
    }
    return(num);
}

```

```

/*****
*
* ENQUEUE *X LOCATION IF LINE OF SIGHT IS POSSIBLE
*
*****/

```

```

*   BETWEEN LAST WAYPOINT IN QUEUE AND *X   *
*
*****/
int lineofsight(QUEUE *q, DATA *x, int *x3, int *y3, int *x4, int *y4)
{
    float r,drx,dry,x1,y1,x2,y2;
    int dr=1,r2,res=0,j,one_not_done = 1,obstacle = 0;
    LINK temp1 = q ->rear;
    if(temp1 != NULL)
    {
        x1 = temp1->d.x;
        y1 = temp1->d.y;
        x2 = x->x;
        y2 = x->y;
        r = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
        r2 = 0;
        if(r!=0)
        {
            drx = dr*(x2-x1)/(float)r;
            dry = dr*(y2-y1)/(float)r;
        }
        while(1)
        {
            for(j=1;j<=precision2;j++)
            if(getbit((int)(x1+j),(int)(y1+j)) == ON ||
                getbit((int)(x1+j),(int)(y1-j)) == ON ||
                getbit((int)(x1-j),(int)(y1+j)) == ON ||
                getbit((int)(x1-j),(int)(y1-j)) == ON )
            {
                if(one_not_done)
                {
                    *x3 = (int)(x1-drx);
                    *y3 = (int)(y1-dry);
                    one_not_done = 0;
                    r2 += dr;
                    x1 += drx;
                    y1 += dry;
                }
                obstacle = 1;
            }
            if(r2>r)
            {
                if(!obstacle)
                    enqueue(q,x);
                return(1+obstacle);
            }
            if(!one_not_done && !obstacle)
            {
                *x4 = (int)x1;
                *y4 = (int)y1;
                return(0);
            }
        }
        obstacle = 0;
    }
}

```

```

        r2 += dr;
        x1 += drx;
        y1 += dry;
    }
}
return(0);
}

```

```

int endtotarget(Queue *q, DATA *x)
{
    float r,drx,dry,x1,y1,x2,y2;
    LINK    temp1 = q->rear;
    if(temp1 != NULL)
    {
        x1 = temp1->d.x;
        y1 = temp1->d.y;
        x2 = x->x;
        y2 = x->y;
        return((int)(sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))));
    }
    return(0);
}

```

```

/*****
 *
 *   OPTIMIZES GENERATED PATH BY ELIMINATING UNNECESSARY
 *   WAYPOINTS.
 *
 *****/

```

```

int optimize(Queue *q)
{
    float drx,dry,r,x1,y1,x2,y2;
    int dr=4,r2,res=0;
    LINK    temp1 = q->front;
    LINK    temp2 = temp1->next;

    gotoxy(1,1);printf("o");
    temp2 = temp2->next;
    if(temp1 != NULL && temp1 != q->rear && temp1->next != q->rear)
    {
        x1 = temp1->d.x;
        y1 = temp1->d.y;
        x2 = temp2->d.x;
        y2 = temp2->d.y;
        r = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
        r2 = 0;
        if(r!=0)
        {
            drx = dr*(x2-x1)/r;
            dry = dr*(y2-y1)/r;

```

```

}
while(1)
{
    int j=precision2;
    if(getbit((int)(x1+j),(int)(y1+j)) == ON ||
        getbit((int)(x1+j),(int)(y1-j)) == ON ||
        getbit((int)(x1-j),(int)(y1+j)) == ON ||
        getbit((int)(x1-j),(int)(y1-j)) == ON )
    {
        temp1 = temp1->next;
        temp2 = temp2->next;
        if(temp2 == NULL)
            return(res);
        x1 = temp1 ->d.x;
        y1 = temp1 ->d.y;
        x2 = temp2 ->d.x;
        y2 = temp2 ->d.y;
        r = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
        r2 = 0;
        if(r!=0)
        {
            drx = dr*(x2-x1)/r;
            dry = dr*(y2-y1)/r;
        }
    }
    r2 += dr;
    x1 += drx;
    y1 += dry;
    if(r2>r)
    {
        farfree(temp1->next);
        temp1->next = temp2;
        temp2->previous = temp1;
        temp2 = temp2->next;
        res = 1;
        if(temp2 == NULL)
            return(res);
        x1 = temp1 ->d.x;
        y1 = temp1 ->d.y;
        x2 = temp2 ->d.x;
        y2 = temp2 ->d.y;
        r = sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
        r2 = 0;
        if(r!=0)
        {
            drx = dr*(x2-x1)/r;
            dry = dr*(y2-y1)/r;
        }
    }
}
}
else
return(res);

```

```
}
```

```
/******  
 *  
 * RETURNS THE FIRST ELEMENT IN QUEUE IN *x AND ERASES *  
 * IT FROM THE QUEUE. *  
 *  
*****/  
void dequeue(QUEUE *q, DATA *x)  
{  
    LINK temp = q -> front;  
  
    if (!isempty(q))  
    {  
        *x = temp -> d;  
        q -> front = temp -> next;  
        q -> front -> previous = NULL;  
        farfree(temp);  
    }  
    else  
        printf("Empty queue.\n");  
}
```

```
/******  
 *  
 * RETURNS THE LAST ELEMENT IN QUEUE IN *x AND ERASES *  
 * IT FROM THE QUEUE. *  
 *  
*****/  
void unqueue(QUEUE *q, DATA *x)  
{  
    LINK temp = q -> rear;  
  
    if (!isempty(q))  
    {  
        q->rear = temp -> previous;  
        q->rear->next = NULL;  
        *x = q->rear->d;  
        farfree(temp);  
    }  
    else  
        printf("Empty queue.\n");  
}
```

```
/******  
 *  
 * ADDES *x TO THE END OF THE QUEUE. *  
 *  
*****
```

```

*****/
void enqueue(Queue *q, DATA *x)
{
    LINK temp;

    gotoxy(1,1);printf("e");
    temp = farmalloc(sizeof(ELEMENT));
    if(temp == NULL)
    {
        printf("NOT ENOUGH MEMORY!");
        exit(1);
    }
    temp->d = *x;
    temp->next = NULL;
    if (isempty(q))
    {
        temp->previous = NULL;
        q->front = q->rear = temp;
    }
    else
    if(abs(temp->d.x-q->rear->d.x)>0 || abs(temp->d.y-q->rear->d.y)>0)
    {
        temp->previous = q->rear;
        q->rear->next = temp;
        q->rear = temp;
    }
    else
        farfree(temp);
}

```

```
/* ZZ_queue.h */
```

```
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;
```

```
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);
```

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

```
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
```

```
*****
```

```
Updates after 22Mar93
```

```
by:       Robert Shanley III
```

```
LAST UPDATE   : 10May94,11May94
```

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

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

```
INCLUDE FILES FOR SUPERVISOR MODULE
```

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

```
#include<alloc.h>  
#include<math.h>  
#include<stdlib.h>  
#include<graphics.h>  
#include<stdio.h>          /*need only for fprintf*/  
#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 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
```

```
extern int graph;  
FILE *ofp;
```



```

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 *);

static double  WayX=1.0,WayY=4.0;
static double  TargetX=1.0,TargetY=4.0;
int  NewTarget=FALSE,Adapt=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=1.0,         /**** GLOBAL X-POSITION ****/
Yposition=4.0,         /**** 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_SendToNav(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;

    ZZ_Real2Screen(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_Supertomapsl(&WayX,&WayY,&NewTarget);

```

```

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");

```

```

}

```

```

/*****
*
* This Function should be called when terminating
* program execution
*
*****/

```

```

void ZZ_EraseSuper(void)
{
    ZZ_CloseGraph();
}

```

```

/*****
*
*   MAIN SUPERVISOR LOOP
*-----*
*
*   The supervisor tasks are divided by function names
*
*****/
#define start 1
void ZZ_SuperLoop(void)
{
    static int FLAG = start;

    ZZ_DrawCar();
    ZZ_DrawCursor();
    ZZ_GetSensorData();
    if(--FLAG<0)
    {
        ZZ_UpdateMap();
        FLAG = start;
    }
    ZZ_DrawCursor();
    ZZ_DrawCar();
    ZZ_ReachTarget();
    ZZ_ReachWayPoint();
    ZZ_CollisionAvoidance();
    {
        int secq;
        double minq;
        secq =60.01*modf(timee/60.01,&minq);
        gotoxy(1,1);printf("TIME : %4.0lf minutes, %2d seconds",minq,secq);
    }
    timee += 0.2l;
    ZZ_SendToNav();
}

```

```

/*****
*
*   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 *
* actually 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++];

    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]);
}

/*****
* SEND ACTUATOR REFERENCE INPUTS TO PROPULSION MODULES. SIGNALS *
* SENT ARE:
*
*   DESIRED STEERING ANGLE
*   DESIRED SPEED
*-----*
*
* When implementing the actuator controls in the
* propulsion module, note that fast response times are
* critical for proper operation of the mobile robot.
*
*****/
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);
}

```

```

/*****
 * THIS FUNCTION DECIDES WHERE MOVING TARGETS ARE LOCATED AND *
 * CHOOSES APPROPRIATE WAYPOINT FOR FUZZY CONTROLLER *
 *-----*
 * NOTE: This function is not yet implemented *
 * *
 *****/
void ZZ_SearchMoving(void)
{
}

```

```

/*****
 * 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.
    ***/
    if(fabs(TargetX-CurrentWaypointX)<TRESHOLDT &&
        fabs(TargetY-CurrentWaypointY)<TRESHOLDT)
    {
        CurrentWaypointX = Xposition;
        CurrentWaypointY = Yposition;
        RoadSpeed=0.0;
        SteeringAngle=0.0;
        Adapt=FALSE;
    }
    /**
    *** 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",0.01,0);
        TargetX = WayX;
    }
}

```



```

    TargetY = WayY;
    timee = 0;
    timer1 = 0;
    METHOD = PLANNER;
    Adapt=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;

/*****
*****
*****      Adaption routines using back propagation
*****
*****/

if(Adapt)
{
int k;
float mu,A,S,alpha=0.00008,tmp,J;
if(WayPointAngle > 180.0)
    A=(WayPointAngle-360)*M_PI/180;
else
    A=WayPointAngle*M_PI/180;
S=(float)SteeringAngle;
if(A > -.780 && A < .780)
{
    ILV[0]=-90;
    for (k=0;k<4;++k)
    {
        ILV[k]=ILV[k]*M_PI/180;
        OLV[k]=OLV[k]*M_PI/180;
    }
    for (k=0;k<4;++k)
    {
        mu=(float)exp(-pow(((A-ILV[k])/ILV[7+k]),2));
        tmp=alpha*(float)(pow(A,2)+pow(S,2))*S;
        OLV[k]=OLV[k]-tmp*mu/SumB;
        ILV[k]=ILV[k]-tmp*(OLV[k]-S)*2*mu*(A-
ILV[k])/(SumB*(float)pow(ILV[7+k],2));
        ILV[7+k]=ILV[7+k]-tmp*(mu-S)*2*mu*(float)pow(A-
ILV[k],2)/(SumB*(float)pow(ILV[7+k],3));
    }
}
}

```

```

for (k=0;k<4;++k)
    {
        ILV[k]=ILV[k]*180/M_PI;
        OLV[k]=OLV[k]*180/M_PI;
    }
ILV[0]=ILV[0]+360;
}
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);

gotoxy(10,5);
printf("OLV= %f %f %f %f",OLV[0],OLV[1],OLV[2],OLV[3]);
gotoxy(10,10);
printf("ILV= %f %f %f %f",ILV[0],ILV[1],ILV[2],ILV[3]);
// gotoxy(10,2);printf("J=%f",J);

}
}

```

```

/*****
*
* 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[]={1,1,1,2,2,2,2,2,3,2,2},
        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,A,R,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]=1*
        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);
}

```



```
/******  
*  
*      Returns Minimum Value Between Two Values      *  
*  
*****/  
double  ZZ_Min(double value1, double value2)  
{  
    if(value1<value2)  
        return(value1);  
    return(value2);  
}
```

```
/* ZZ_supr2.h */

#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);
```

```

/*****
FILE      : ZZ_VSION.C

DESCRIPTION  : VISION MODULE FOR AUTONOMOUS VEHICLE

                This file contains code for SONAR SIMULATION,
                CAN COMMUNICATIONS, high level reasoning.
-----

by        : Ricardo Andujar

LAST UPDATE   : MARCH 22, 1993
*****/

```

```

/*****
INCLUDE FILES FOR VISION MODULE
*****/
#include<alloc.h>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
#include"ZZ_GRAPH.H"
#include"ZZ_OBJECT.H"
#include"ZZ_CARSP.H"
#include"ZZ_CAN.H"
#include"ZZ_VSION.H"

```

```

/***** PRIVATE IDENTIFIERS *****/

```

```

/*****
***
*** The following external declaration is needed for the
*** sonar simulation only. It should be eliminated when
*** simulation is no longer needed !
***
*****/
extern ZZ_NavToVision(long *,long *,long *);

```

```

extern int graph;

int ZZ_Circle = 0;

```

```
/****** PRIVATE DECLARATIONS *****/
```

```
static double
```

```
    Xc[NUM_SENSORS],      /*** X SENSOR POSITIONS (pixels) ***/  
    Yc[NUM_SENSORS],      /*** Y SENSOR POSITIONS (pixels) ***/  
    Rxy[NUM_SENSORS],     /*** SENSOR RADIUS FROM REFERENCE ***/  
    AngleXY[NUM_SENSORS], /*** SENSOR ANGLE FROM REF. POINT ***/  
    Range[NUM_SENSORS],  
    Range2[NUM_SENSORS],  
    Compass,  
    *Compass2,  
    *Xposition,  
    *Yposition,  
    RoadSpeed,  
    SteerAngle;
```

```
static int  EMERGENCY,  
           Xconvert,  
           Yconvert;
```

```
static void ZZ_UpdateSensorMeas(void);  
static void ZZ_SonarSim(void);  
static void ZZ_TransmitSensorMeas(void);
```

```
/******  
 *  
 * CALL THIS FUNCTION WHEN INITIALIZING THE VISION MODULE *  
 *  
 *****/
```

```
void ZZ_InitVision(void)
```

```
{  
    int i;
```

```
/******  
 ***  
 *** THE FOLLOWING STATEMENTS IS FOR SONAR SIMULATION  
 *** PURPOSES ONLY. WHEN FINALLY IMPLEMENTED THE FOLLOWING  
 *** STATEMENT SHOULD BE ERASED !  
 ***  
 *****/
```

```
    for(i=0;i<NUM_SENSORS;i++)  
        ZZ_Cart2Polar(Xr[i],Yr[i],&AngleXY[i],&Rxy[i]);  
    ZZ_NavToVision(&Xposition,&Yposition,&Compass2);  
}
```

```

/*****
*
*   MAIN VISION LOOP
*
*****/
void ZZ_VisionLoop(void)
{
    ZZ_UpdateSensorMeas();
}

/*****
*
*   SONAR SIMULATION
*-----*
*   Sonar Range Information is received with a delay proportional *
*   to the measured distance.
*
*****/
void ZZ_SonarSim(void)
{
    int i,Rangep[NUM_SENSORS],radius;
    double j,Delta;

    ZZ_Real2Screen(*Xposition,*Yposition,&Xconvert,&Yconvert);

    for(i=0;i<NUM_SENSORS;i++)
        ZZ_Polar2Cart(AngleXY[i]+*Compass2,
            Rxy[i]*SCREENCONVERT,&Xc[i],&Yc[i]);

    for(i=0;i<NUM_SENSORS;i++)
    {
        float anglesin1 = sin(-*Compass2-NominalAngle[i]
            +HALFSPREADANGLE),
            anglecos1 = cos(-*Compass2-NominalAngle[i]
            +HALFSPREADANGLE),
            deltasin=(sin(-*Compass2-NominalAngle[i]
            -HALFSPREADANGLE)-anglesin1)/5.0,
            deltacos=(cos(-*Compass2-NominalAngle[i]
            -HALFSPREADANGLE)-anglecos1)/5.0;

        int j,
            xtemp=Xc[i]+Xconvert,
            ytemp=Yc[i]+Yconvert;
    }
}

```

```

radius = 0;
Rangep[i] = 0;

if(ZZ_Circle) ZZ_DrawCircle();

while(1)
{
    for(j=0;j<=5;j++)
    {
        Rangep[i] = getpixel(
            xtemp-radius*(anglesin1+j*deltasin),
            ytemp-radius*(anglecos1+j*deltacos));

        if(Rangep[i]==WHITE || Rangep[i]==LIGHTRED
            || radius > SMAXRANGE)
            break;
    }
    radius +=3;

/*****
*
*   Limit Sensor Range
*
*****/
    if(radius>SMAXRANGE)
    {
        radius = SMAXRANGE;
        Range[i] = radius/SCREENCONVERT;
        if(ZZ_Circle) ZZ_EraseCircle();
        break;
    }

/*****
*
*   Limit Sensor Range
*
*****/
    if(Rangep[i]==WHITE || Rangep[i]==LIGHTRED)
    {
        radius -= 7;
        if(radius<0)radius = 2;
        Range[i] = radius/SCREENCONVERT;
        if(ZZ_Circle) ZZ_EraseCircle();
        break;
    }
}
for(i=0;i<NUM_SENSORS;i++)
    Range2[i] = Range[i];
}

```

```

/*****
 *
 *   SONAR SENSOR MEASUREMENT GOES HERE
 *
 *****/
void ZZ_UpdateSensorMeas(void)
{
/*****
 *
 *   Following function to be replaced with data acquisition
 *
 *****/
    ZZ_SonarSim();

/*****
 *
 *   IF Emergency is enabled, monitor range information for
 *   Emergency Collision Avoidance
 *
 *****/
    if(EMERGENCY)
    {
/*****
 *
 *   Request Sonar Sensor Range Information
 *
 *****/
        ZZ_CAN_Request(HIGHPRIORITY,VISION,NAVIGATION,&DataContent,
                        Data,&DataNum);

        RoadSpeed = Data[1];
        SteerAngle = Data[2];

/*****
 *
 *   INSTINCT BEHAVIOR
 *
 *****/
        if ((Range[1]<RoadSpeed*ROADRANGEC1) ||

```

```

        (Range[2]<RoadSpeed*ROADRANGEC2))
    {
        Data[1] = 0;
        Data[2] = 0;
        Data[3] = ON;
        ZZ_CAN_Command(HIGHPRIORITY,VISION,NAVIGATION,
            &DataContent,Data,&DataNum);
    }
}

```

```

/*****
*
*   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;
    }
}

```

```

/*****
*
*   CALL THIS FUNCTION WHEN QUITTING PROGRAM
*
*****/

```

```

void ZZ_EraseVision(void)
{
}

```



```
/*ZZ_vision.h */

#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 HALFSREADANGLE 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);
```

VITA

Robert L. Shanley III

Candidate for the Degree of

Master of Science

Thesis: ENVIRONMENT MAPPING AND ADAPTIVE FUZZY LOGIC CONTROL
FOR AN AUTONOMOUS VEHICLE

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in West Lafayette Indiana, April 20, 1969, the son of Robert L. Shanley II and Mary Ann Shanley.

Education: Graduated from North Central High School, Indianapolis, Indiana, in May 1987; received Bachelor of Science Degree in Aerospace Engineering from Purdue University at West Lafayette Indiana in December 1992; completed requirements for the Master of Science degree at Oklahoma State University in July, 1994.

Professional Experience: Teaching Assistant, Department of Mechanical and Aerospace Engineering, Oklahoma State University, January, 1993, to May, 1994.

Cooperative education program at the Phillips Laboratory, Edwards AFB, January 1989, to December 1990.