AUTONOMOUS VEHICLE CONTROL USING FUZZY INFERENCE AND A FAST PATH PLANNING ALGORITHM

By

RICARDO ANDUJAR

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1991

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, 1993

OKLAHOMA STATE UNIVERSITY

AUTONOMOUS VEHICLE CONTROL USING FUZZY INFERENCE AND A FAST PATH PLANNING ALGORITHM

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 and undergraduate 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 Mike Moan who saved me hours of literature review. Thanks go to him for helping me find very important research information.

To Jim Hartwig, who's friendship and advice is greatly appreciated. To Robert Taylor and Jeff Lopez, who cheered me up during those long study sessions, thanks go to them for keeping me up all night.

To my parents, Ricardo Andujar Sr. and Felipa Gutierrez, 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 the most important person in my life and soon to be wife, Ana D. Chavarria, who loves with all her heart. She made me feel special and kept me on my toes. Life would not be the same without her.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Autonomous Car-Like Robot in Unstructured Dynamic Environments

A problem commonly encountered when combining planning and mapping algorithms using sonar range information to implement an autonomous car-like mobile robot is that computational overhead will increase due to incompatibility between the path planner and the mapping algorithm. This is especially true when an autonomous robot in an unknown dynamic environment is implemented. This increased computational overhead degrades the real-time operation of the robot. The reason is that many of the planning algorithms were developed independently from mapping algorithms, or dynamic environments were not considered in mapping algorithms. Some path planners did not consider the kinematic limitations of the car-like robot, see [4] and [5], thus generating some paths that were physically impossible to execute by the robot. The solution to these problems was to concurrently design the path planner and mapping algorithm for optimized real-time operation while considering all the kinematic constraints of the mobile robot.

The aim of the author was to develop a supervisor controller for an autonomous car-like mobile robot in unstructured environments that includes a high level of reactivity and fast complex path-planning capabilities. Continuous adaptive mapping of the environment would allow the robot to detect and map moving objects in real-time with the capability to generate paths around them or to track them, depending on the assigned task.

The supervisor controller must conform to a controller area network architecture to interact with the other components of the robot, the vision and propulsion modules. The new path planning algorithm designed consists of both a sensor based path planner and a mapped path planner. The sensor based path planner is variation of the one developed by Huang and Lee [10], since the robot used has kinematic limitations. The mapped path planner is also derived from the algorithm developed by Huang and Lee [10]. This path planner is very fast, generating complex sub-optimal paths in 1 second or less. A one plane bit map is used to generate and update the map of the environment. The supervisor controller uses fuzzy inference to execute the generated path and simultaneously avoid collisions with stationary obstacles.

Simulation results are presented and discussed in chapter vi, where optimal real-time operation is the decisive factor in the efficiency of the supervisor module.

Potential Applications

Mobile robots are currently used in manufacturing plants, hospitals, and for toxic waste disposal. These robots offer some degree of flexibility, but still there is need for robots which can adapt to a unknown and changing environment and make appropriate decisions without human supervision. The area where this is needed most is in toxic environments, where human supervision is dangerous and sometimes fatal. The robot controller currently being developed can also be used for manufacturing plants where the floor plan is constantly being changed. Magnetic stripes or grooves used to guide mobile robots in manufacturing environments can be eliminated since the robot being developed is constantly updating the environment map.

CHAPTER II

LITERATURE REVIEW

Controller Area Network

The autonomous car-like mobile robot devised will use a controller area network (CAN) architecture, where the components of the robot function separately. The J1939 committee of the Society of Agricultural Engineers (SAE) is currently working on a standard framework for a CAN [1]. The author will follow SAE's recommended practices for the simulation and implementation of the CAN.

Autonomous Mobile Robot Architecture

The generic architecture for the autonomous mobile robot used by the author was proposed by Noreils and Prajoux [2]. It was developed with integration of reflexive and planning capabilities, while providing important features such as : progressive and programmable reactivity, robustness, an versatility. The generic architecture they proposed is composed of three levels: functional, control, and planner. The author's implementation of this robot architecture does not include 2-D camera or laser range finders, it functions strictly with sonar for navigation.

Mobile Robot Navigation

A method for navigation of a car-like mobile robot is presented by Vasseur, Pin, and Taylor [3]. The method is computationally efficient involving the decomposition of the environment in convex cells. When continuously updating and adapting the environment map the computational overhead increases due to the decomposition of the environment in convex cells at every control step. Vasseur, Pin, and Taylor are currently working on a sensor based navigation version of this method. Another computationally efficient algorithm for navigation was presented by Zelinsky [4]. This suffers from the same computational overhead increase when implemented with continuous remapping. This is due to recalculation of the quadtree partitioning at every step. The author's implementation of the path planner is a variation and adaptation to the general method developed by Huang and Lee [10]. A sensor and a mapped path planner is implemented. The sensor based version is used when there is no information of the environment, while the mapped version is used when environment information is available.

Collision Avoidance for Mobile Robots

Methods for collision avoidance of the mobile robot are presented by Holenstein and Badreddin [5] and Krause and Zimmerman [6]. Holenstein and Badreddin use an approach similar to artificial potential field approach (Khatib [7] and Krogh [8]). They build the potential-field around the robot body instead of around the obstacles themselves. This algorithm is suited for a static environment but modifications are necessary when dynamic environments are considered. A fuzzy logic controller with object avoidance is implemented by Krause and Zimmerman. Their controller was designed for an outdoor high speed robot with three sonar sensors with maximum operating speeds between 20 to 30 mph, indicating a high level of reactivity. Due to the exceptional results obtained by Krause and Zimmerman and the flexibility of a fuzzy logic controller, an implementation using a

fuzzy controller with dynamic collision avoidance was chosen over the potential-field approach. The design of the controller was designed taking into consideration the design of the path planner

Mobile Robot Mapping in Unstructured Environments

Treatment of Systematic errors in the processing of wide angle sonar sensor data for robotic navigation was considered by Beckerman and Oblow [9]. They proposed a 4 valued labeling scheme to resolve conflicting sonar data when updating the navigation map. Since navigation is the main use for the mapping algorithm, the necessary implementation used is a simplified version of Beckerman and Oblow's method by skipping the intermediate phase and directly using a binary labeling scheme.

CHAPTER III

INTELLIGENT CONTROLLER USING CONTROLLER AREA NETWORK

Controller Area Network Architecture

The robot will be using three independent computer boards communicating through a communications network tailored specifically for control applications. The CAN architecture that will be followed in simulating and implementing the mobile robot is the standard architecture being drafted (J1939) by the society of agricultural engineers as of January 1993 [1]. The following information is a short summary of the specification' s for all the components of the CAN. For more detailed information see [5]. Figure 1 shows a general setup for a controller area network.

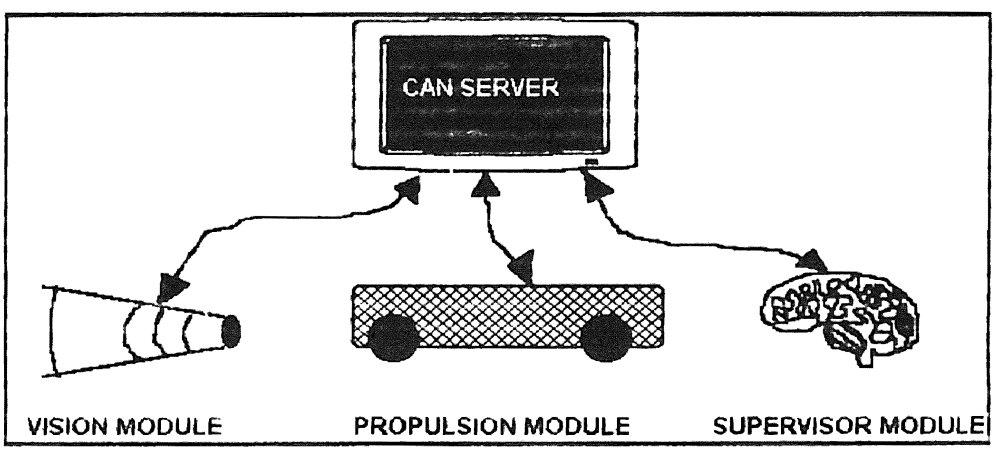

Figure 1. General Setup for Controller Area Network.

SAE Recommended Practice for Serial Control and Communications Network (Class C) for Truck and Bus Applications

Application Layer. Although this standard CAN architecture was intended for heavy duty truck and bus vehicle use, it can be utilized by other applications. The mobile robot will only use a fraction of the capabilities of the Network.

Physical Layer. The Data Transmission Rate for the Network is 250K bits/sec using a twisted shielded pair. The physical layer is a realization of an electrical connection of a number of ECUs (Electronic Control Unit) to a network. The total number of ECU's will be limited by electrical loads on the bus line. This maximum number of ECU's is fixed to 30 due to the definition of the electrical parameters given in the present specification.

Data Link Layer. The data link layer provides for the reliable transfer of data across the physical link. This consists of sending the CAN message frame with the necessary synchronization, error control and flow control. The flow control is accomplished by a consistent message frame format. The CAN specification referenced is "CAN Specification 2.0 Part B", September 1991. J1939 contains additional requirements that are not specified by CAN. Figures 2 and 3 show the CAN Standard and Extended Frame Format.

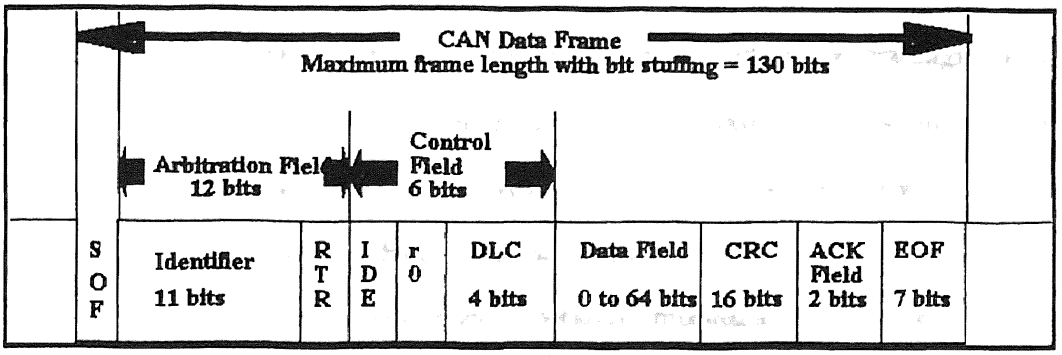

Figure 2. CAN Standard Frame Format

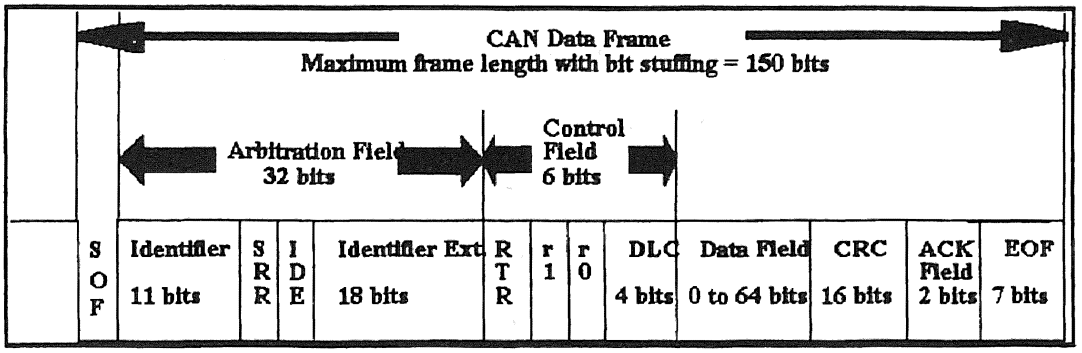

Figure 3. CAN Extended Frame Format

Network Management Protocol. The primary functions assigned to the Network Management protocol are those of Address Management and Network Error Management. Address Management consists of those functions that recognize, allocate addresses to, and monitor the existence of nodes on a given network. These functions provide for the association of a device type identification with a given node address or address block on the network.

Supervisor Module

All the long term planning and decision processes are handled by the supervisor module. The supervisor uses the network to request information from the vision and

propulsion module to obtain sonar range information and robot sensor readings, including position and bearing, respectively. When the supervisor module first goes on-line it must send initial position and heading to propulsion module, then it will subsequently rely on the position and heading updated by the propulsion module.

The following chapter contains more detailed information for the supervisor module.

Vision Module

The vision module updates sonar range information, reads current speed and steering angle, and monitors proximity to surrounding obstacles. In case of danger of collision, the vision module gains temporarily control of the propulsion module and steers the robot away from danger. The vision handles any requests from the supervisor module, and monitors supervisor status in case of failure. If supervisor module becomes inoperational the vision module must place the robot in a safe location and command the propulsion module to stop.

Propulsion Module

The propulsion module updates propulsion actuators and maintains the current robot location and bearing. The supervisor module is allowed to request information and set reference control signals to the propulsion module. In case of an emergency, the vision module will temporarily gain control of the propulsion module, that is, supervisor commands will be ignored until the emergency situation is remedied by the vision module. An emergency enable/disable feature is available to the supervisor to prevent the vision module form taking control of the propulsion module. When initializing, the propulsion module must request initial bearing and position from the supervisor module. If the supervisor is not yet on-line, the propulsion will continue requesting the said information

until the supervisor logs on the network. In case of failure of the supervisor or vision module the propulsion module must stop immediately, and wait for further instructions from either vision for propulsion module.

CHAPTER IV

SUPERVISOR MODULE

The simulation model assumptions are detailed in Appendix A.

Fast Sub-Optimal Path Planning

An improvement to the sensor based path planner algorithm developed by Huang and Lee is discussed. When no environment information is available, a variation to Huang's algorithm is used, which includes fuzzy collision avoidance, and changes required to accommodate the kinematic restrictions of the robot used. The upper bound of the total path length is given by Huang and Lee [10],

$$D \leq f_H + \sum_{n=1}^{N} k_n B_n \tag{1}$$

where f_H is the total path length generated by the Heuristic-mode. The second term on the right hand side is the contribution of the Tracking -mode. From Huang and Lee [10].The modified algorithm used for the sensor based algorithm is as follows:

1) Initialization, i=1, current position is the starting position S.

2) Heuristic-mode: From current position proceed toward target by fuzzy inference until one of the following occurs. (Collision avoidance algorithm takes eliminates the need to generate median points.)

   a) Target is reached. Stop.

   b) The robot gets stuck in a concave obstacle. Then the current point is defined as a hit point. Go to Step 3.

3) Tracking - mode: Follow the contour of the obstacle boundary until the following occurs.

a) Target is reached. Stop.

b) Define a leave point L_i that satisfies $\angle H_i L_i T = 180^\circ$, where H_i is the current hit point and T is the target. Go to Step 2.

c) The robot returns to the last hit point. No obstacle-free path exists. Stop.

Once the environment has been mapped, the mapped version of the path planner is used, again based on a variation of the method developed by Huang. One of the improvements over the original algorithm is the fact that the collision avoidance operates almost independently from the path planner, thus increasing the reactivity while retaining planning capabilities. Since the path is generated beforehand and is optimized to the minimum distance, the upper bound of the total path length using the mapped version will be much smaller. In practice, this is true in the great majority of cases, but there are instances when this will not occur, due to conflicts generated by the collision algorithm. Intermediate goals called way points are generated by the mapped path planner to facilitate the actual path followed by the robot. Way points are defined at the edges of two straight lines of the path. Each way point is stored in a queue to be used later when executing the generated path. The second phase of the mapped path planner consists of eliminating unnecessary way points by testing the following logic. If a line can be generated between two non adjacent way points without crossing an obstacles than eliminate all the way points that are between the test way points. Figures 4 - 5 demonstrate the use of this logic.

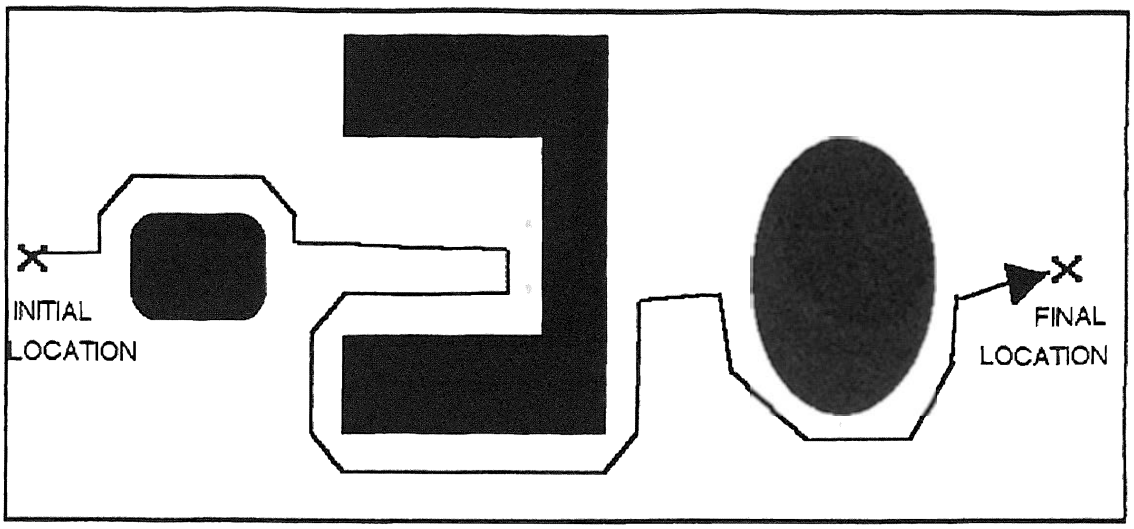

Figure 4. Phase One - Basic Path Planning

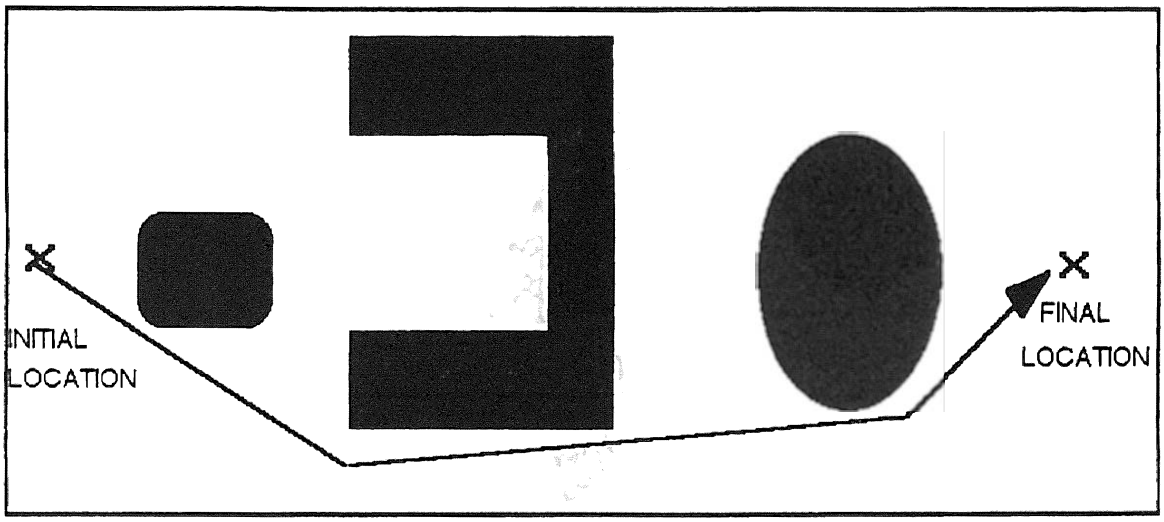

Figure 5. Phase Two - Optimized Path Planning

From the path generated, the path was optimized by eliminating the unnecessary lines used to reach the target.

Map Creation and Updating

Initially, all map space is assumed to be occupied space. Every time the supervisor obtains sonar range information and current position and bearing, the map is updated by filling a triangular polygon representing the space covered by each sonar sensor. Current bearing and position of the robot is critical in order to draw the appropriate triangular polygon for each sonar sensor on the environment map. Without it, it is impossible to determine where the robot is, unless other visual inputs are used. The robot simulation uses six sonar to map the environment. Figure 6 shows an example of this configuration.

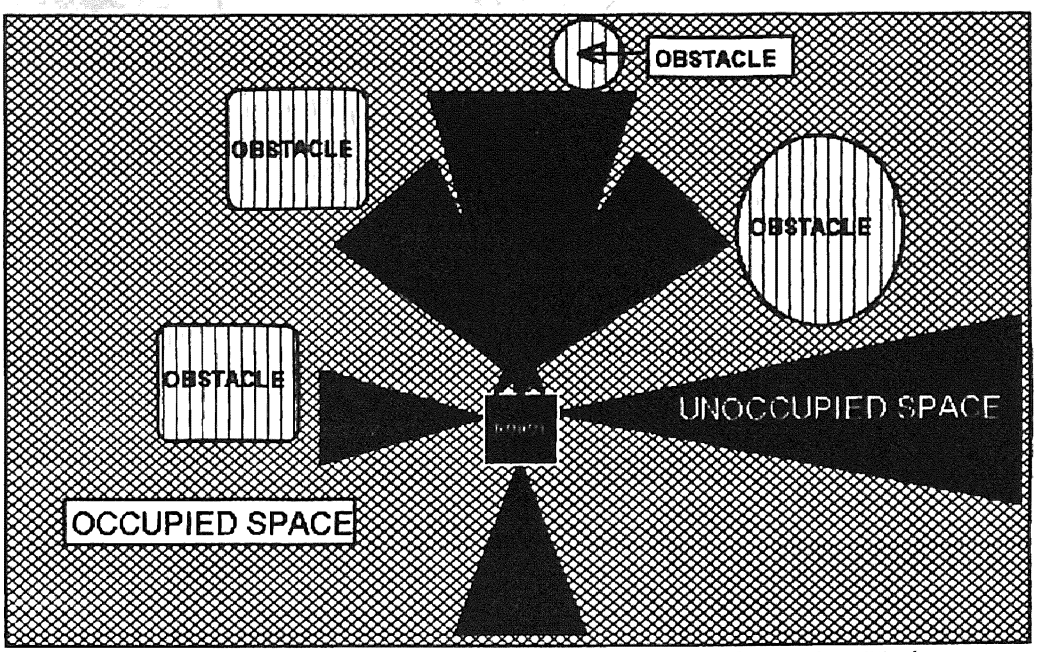

Figure 6. Initial Map Creation With First Sonar Information Update

If a new object is located in a region of the map previously mapped as unoccupied space, the procedure is to place a patch of occupied space following the sonar reading. The logic used to detect new or moving objects is to search the area surrounding the points

that are at the edge of the current sonar scan. On the next page figure 7 shows an example of a occupied patch being placed over a new obstacle. This also can be used to detect moving objects, or one can track the position of a moving object by placing the final location as the center of the last patch placed. The only problem that arises is that there can be more than one moving obstacle, thus more logic is required to decide which obstacle to track.

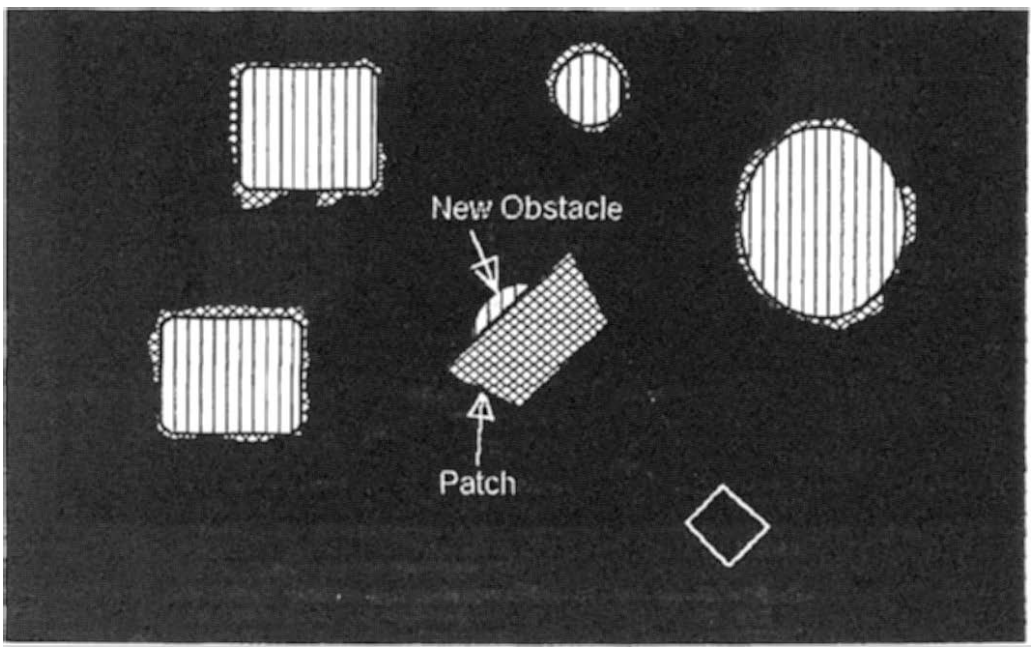

Figure 7. Placing Patch When New Obstacle Detected

Updating the environment map using triangular polygons and occupied patches can reduce small amounts of drift due to global positioning and heading inaccuracies by placing patches of occupied space on occupied parts of map that are on unoccupied parts and erasing occupied space that is now unoccupied space. Eventually the map will drift away from the confines of the memory allocated for it and necessary adjustments must be

made. By having the robot associate an x-y location with a reference region and then providing a periodic homing function drift position and heading drift can be minimized. The mobile robot simulation used assumes no position and heading drift, the focus of this paper was to develop and test a supervisor module suitable for real-time application in unknown dynamic environments. Position and heading drift corrections will be dealt with during the implementation of the mobile robot once the nature of the errors are investigated.

Fuzzy Logic Control And Collision Avoidance

The fuzzy inference rule that is used to implement the fuzzy controller is the MAX/MIN rule. The defuzzification method used is Larsen's Rule and using the mean of the output fuzzy variables. For more information see [11].

The fuzzy logic controller is divided in two parts. The first controller adjusts the speed and steering angle in order to reach the current way point supplied by the path planner. The second controller adjusts the steering angle and speed computed from the first controller to get out of tight spots and avoid collisions with static and moving obstacles. Adjustments to the steering angle and speed are only made when necessary, otherwise the steering angle and speed sent to propulsion module is unchanged from the first controller. 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 8 to figure 11 show membership functions for all the linguistic variables used in the first controller.

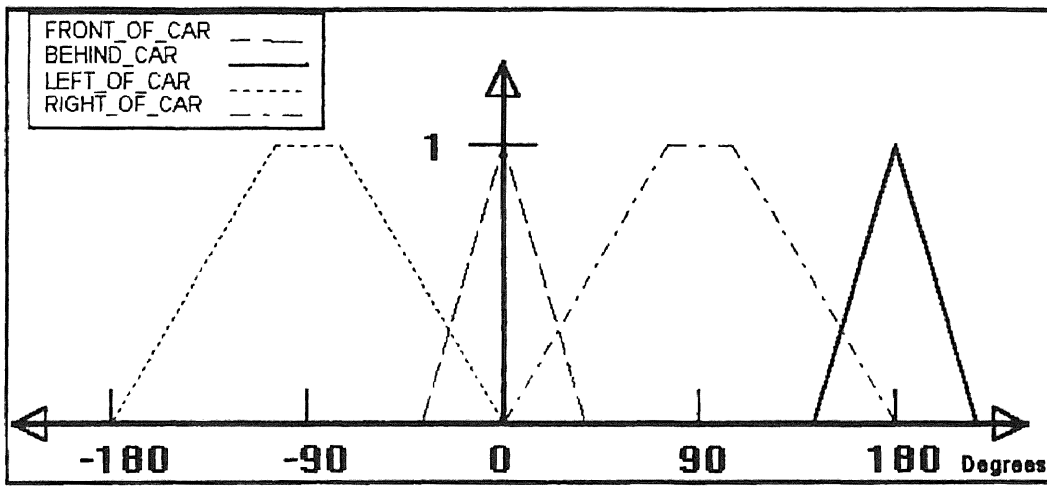

Figure 8. Location of Waypoint Relative to Robot

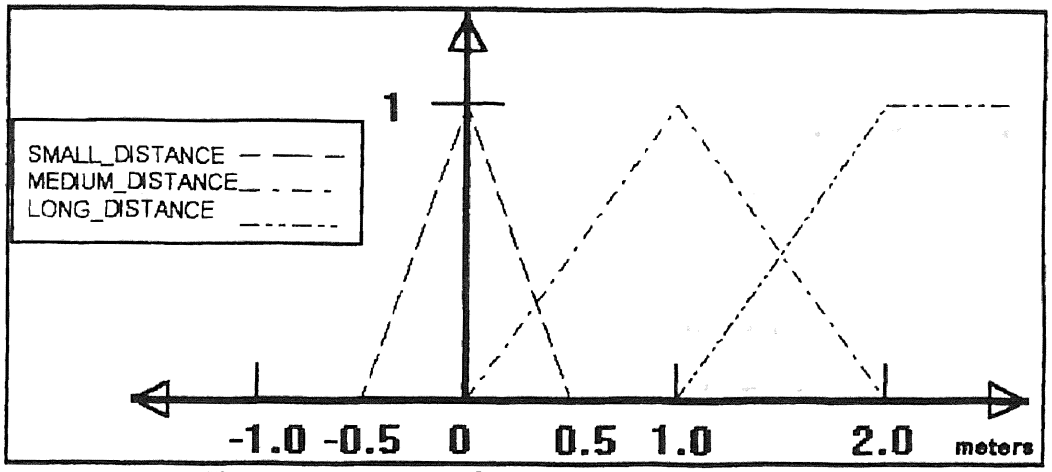

Figure 9. Distance of Waypoint Relative to Robot

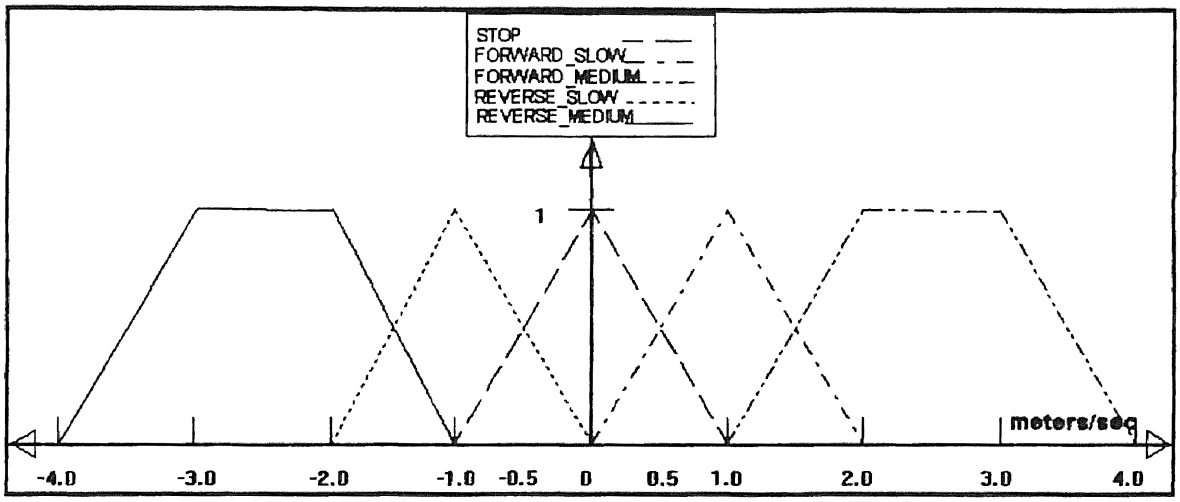

Figure 10. Desired Speed of Robot

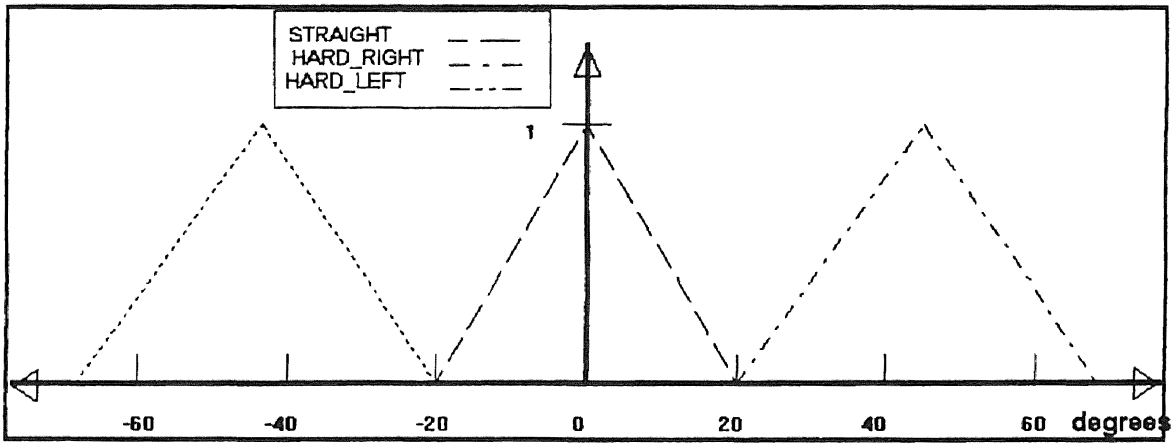

Figure 11. Desired Steering Angle of Robot

Triangular membership functions were sufficient for almost all linguistic variables in computing the nominal steering angle and desired speed. Trapezoidal membership functions were required for LEFT_OF_CAR and RIGHT_OF_CAR to avoid some spots where the computed speed was zero even though the current waypoint had not been reached. FORWARD_MEDIUM and REVERSE_MEDIUM have a trapezoidal shape so as the the robot approaches the current waypoint the speed will decrease at a faster rate the closer it gets to the waypoint.

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 12 through figure 14 on the next page show membership functions for all the linguistic variables used in the second controller.

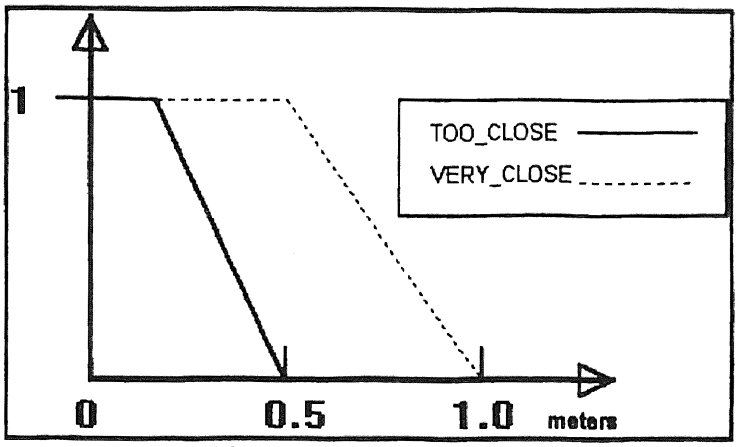

Figure 12. Sensor Distance to Obstacle Membership Functions

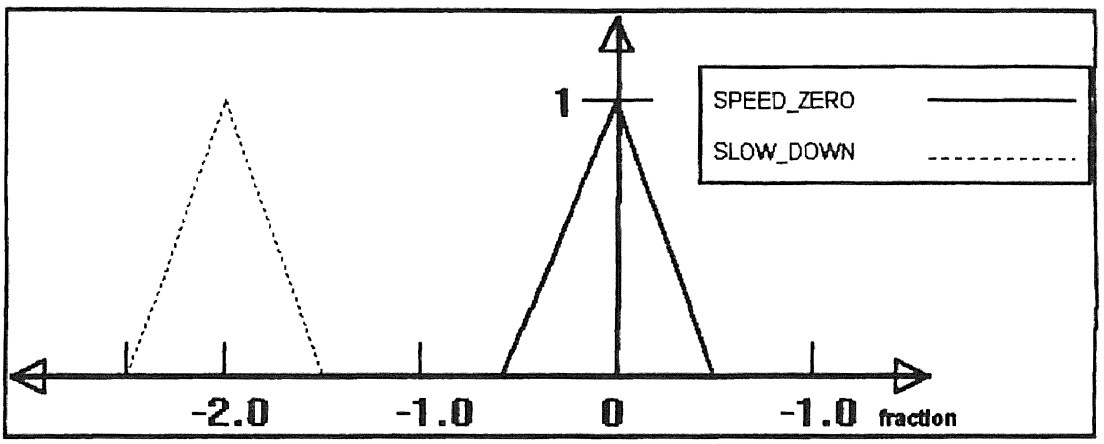

Figure 13. Speed Change Membership Functions

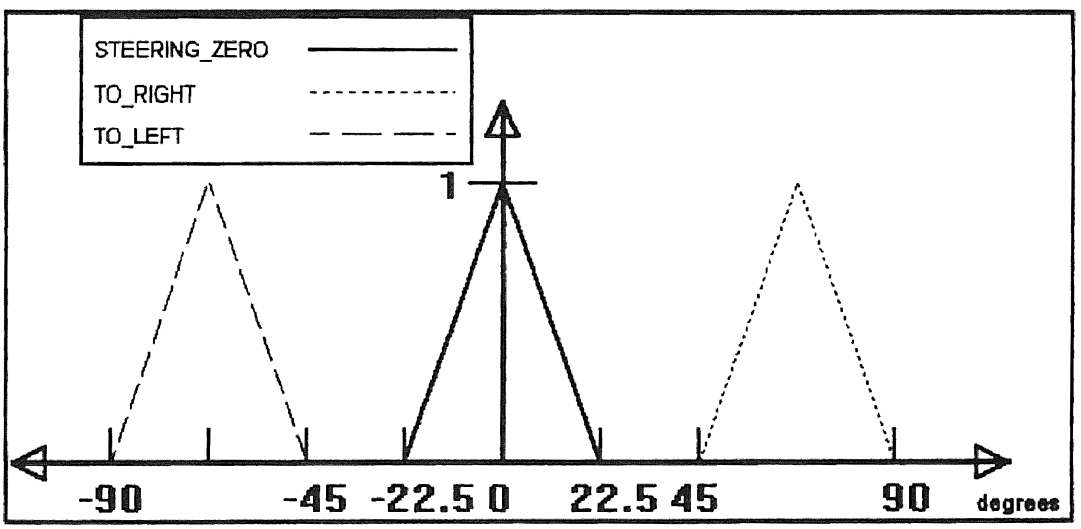

Figure 14. Steering Change Membership Functions

CHAPTER V

THE ROBOT SIMULATION SOFTWARE

This chapter is an overview of the software functionality. For more detailed information see the simulation model assumptions in Appendix A and the software code in Appendix B.

The Supervisor Module Software

This is where all the path planning, mapping, and collision avoidance is done. The supervisor module consists of using the sonar range information to update the environment map, avoid collision, and reach a determined target. Sampling rate for the supervisor module is 5Hz.

The Vision Module Software

This module updates the sonar range information and provides a backup collision avoidance independently from the supervisor module. Whenever it decides to control the robot it will send steering angle and desired speed information to the propulsion module.

The vision module sonar simulation consists of scanning the sector where the actual sonar signal would occupy. Initially, the scan starts at the smallest radius allowed by the pixel screen. The scan proceeds from the initial angle of the sector to the final angle. If an obstacle was encountered, a pixel designated as obstacle, then the current radius is used as the range obtained by that sensor, otherwise, the radius is increased and the scan is started over.

The Propulsion Module Software

The actuator control and global positioning is accomplished in this module. The network server function handles all requests from the supervisor and vision module. Currently the global positioning is approximated using velocity and heading information. Sampling rate for the controller is 20 Hz. The DC-motors are modeled and controlled using PI-controllers.

The Controller Area Network Simulator

The controller area network simulator is used to establish the format that will be used throughout all software code generation in all the modules. Currently there are three functions implemented. They are used to install each module as a terminal on the network, communication requests, and send commands to other modules through the network. Since the CAN is not the focus of this paper, the current implementation is very simple in nature.

The Fuzzy Set Library

The fuzzy set library is a collection of functions developed to facilitate the construction of fuzzy inference rules. A brief explanation for each function is provided.

Membership Functions

The create trapezoidal membership functions there is a declared type as follows:

```
typedef  struct ZZ_TRAPEZOID
{
        float    bottomleft,topleft,topright,bottomright;
};
```

To create trapezoidal or triangular membership functions just define a variable using the ZZ_TRAPEZOID data type and define your membership limits.

Example:

```
ZZ_TRAPEZOIDAL  medium={0,0.1,0.2,0.3};
```

If bottomleft>topleft than the membership function will be 1 for values smaller than topleft. If bottomright<topright than the membership function will be 1 for values larger then topright.

Membership Function Values for Singleton Inputs

Returns the degree of truth of a given membership for a given value. The function prototype is

```
double   ZZ_Member(value, ZZ_TRAPEZOID *shape);
```

Maximum Value Between Two Numbers

The function returns the higher number between the two numbers provided to the function. The function prototype is

```
double          ZZ_Max(double value1, double value2);
```

Minimum Value Between Two Numbers

This function returns the lowest value between the two numbers provided to the function. The function prototype is

```
double          ZZ_Min(double value1, double value2);
```

Complement of Membership Functions

This function returns the complement of the membership function provided. The function prototype is,

```
double    ZZ_Complement(double(double value, ZZ_TRAPEZOID *shape);
```

Create a Discrete Fuzzy Output Variable

Allocates memory for a discrete fuzzy output variable and sets the range and number of discrete points. The function prototype is,

```
double ZZ_InitFuzzyOutput(double lowrange, double highrange,
                          ZZ_FUZZYOUTPUT  *funct, int discretenum);
```

Larsen's Rule

This function compares the trapezoidal membership function with the membership values stored in the discrete fuzzy output. The function prototype is,

```
void    ZZ_AddMax(double min, ZZ_TRAPEZOID *memb,
                  ZZ_FUZZYOUTPUT *funct);
```

Deffuzification

This function defuzzifies using the mean value. The function prototype is,

```
double ZZ_Defuzzify(ZZ_FUZZYOUTPUT   *funct);
```

Resetting Discrete Fuzzy Number

This function resets all the values in the discrete fuzzy number to zero. The function prototype is,

```
void    ZZ_ClearFuzzyOutput(ZZ_FUZZYOUTPUT *funct);
```

CHAPTER VI

SIMULATION RESULTS AND DISCUSSION

The following examples will be tested and discussed in detail using the proposed path planner, adaptive environment mapper, and fuzzy controller.

1. Robot will reach two target locations while avoiding five static obstacles. Environment is unknown at starting time. Only initial and target location are known.
2. Robot will reach two target locations while avoiding ten static obstacles. Environment is unknown at starting time. Only initial and target location are known.
3. Robot will reach a target located in a different room while avoiding various static obstacles. Environment is unknown at starting time. Only initial and target location are known.

The simulated environments used for examples 1 through 3 are in figures 15 through 17, shown on the following two pages. The first target is located using the sensor based path planner while the path to the second target is generated using the current map of the environment. Note again that the mapping algorithm uses range information obtained from the sonar simulation. For detailed information refer to vision module software, chapter V.

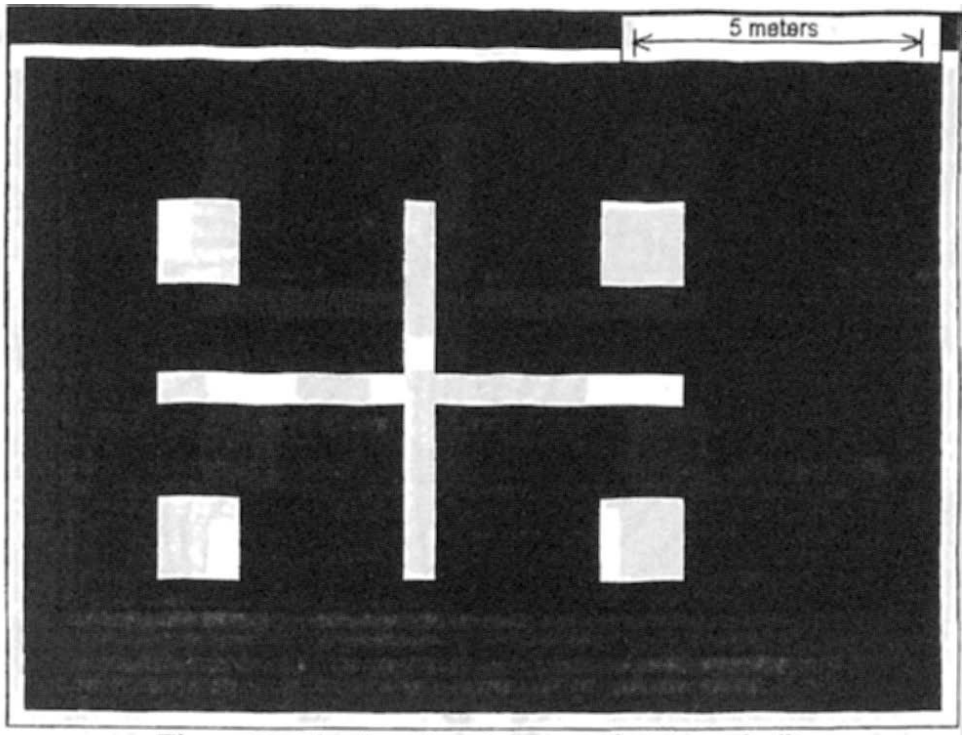

Figure 15. Simulated Environment for Example #1

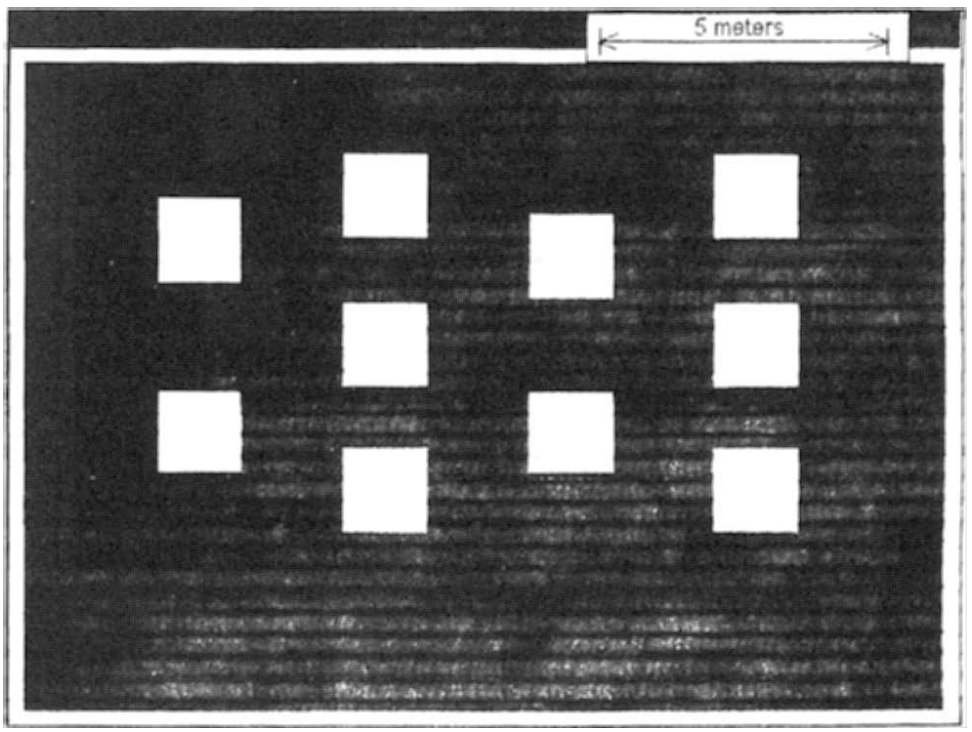

Figure 16. Simulated Environment for Example #2.

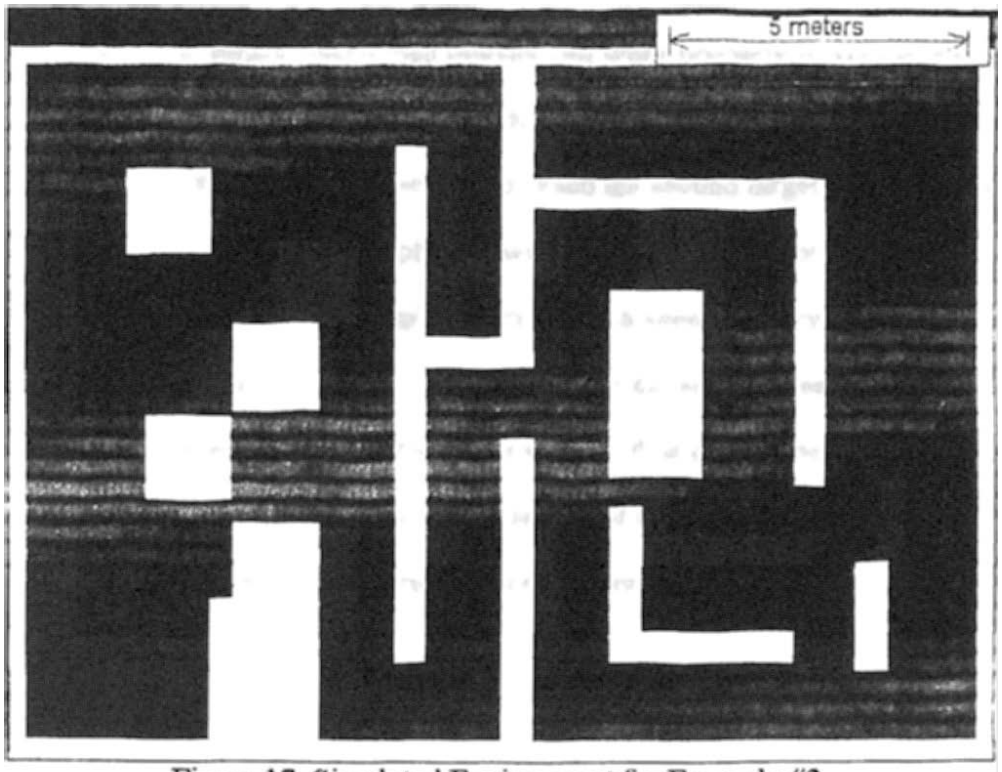

Figure 17. Simulated Environment for Example #3.

Simulation Example #1

The robot succeeded in reaching its first destination across the room. Figures 18 through 23, on the following four pages, show the time progression of the robot through the environment, first using a sensor based path planner, followed by the mapped path planner. The first target was located after 19 minutes, 5 seconds. The reason is that the sensor based algorithm used does not function very well when working around concave obstacles, such as in this simulation. When encountering a concave obstacle, the implemented robot sensor based planner algorithm used will take more time to execute, the robot will is forced to backup more times, than if a convex obstacle would have been encountered. A improved version of the sensor based algorithm is currently being developed. New fuzzy inference rules can be added to the existing controller rules to remedy part of the problem. The

current version of the sensor based path planner did work properly in almost every situations, and served its purpose for this study.

The robot required only 59 seconds to reach the second target, which was at its initial location. The map based path planner was much more efficient than the sensor based planner. The time required to generate the path was 0.6 seconds, very fast compared to potential field path planners. A good representation of the environment also helped to reduce the time required to generate the required path, thus generating an optimized path to the target. This example demonstrates the advantage of a mapped path planner vs. a sensor based path planner. No problems were encountered during the simulation.

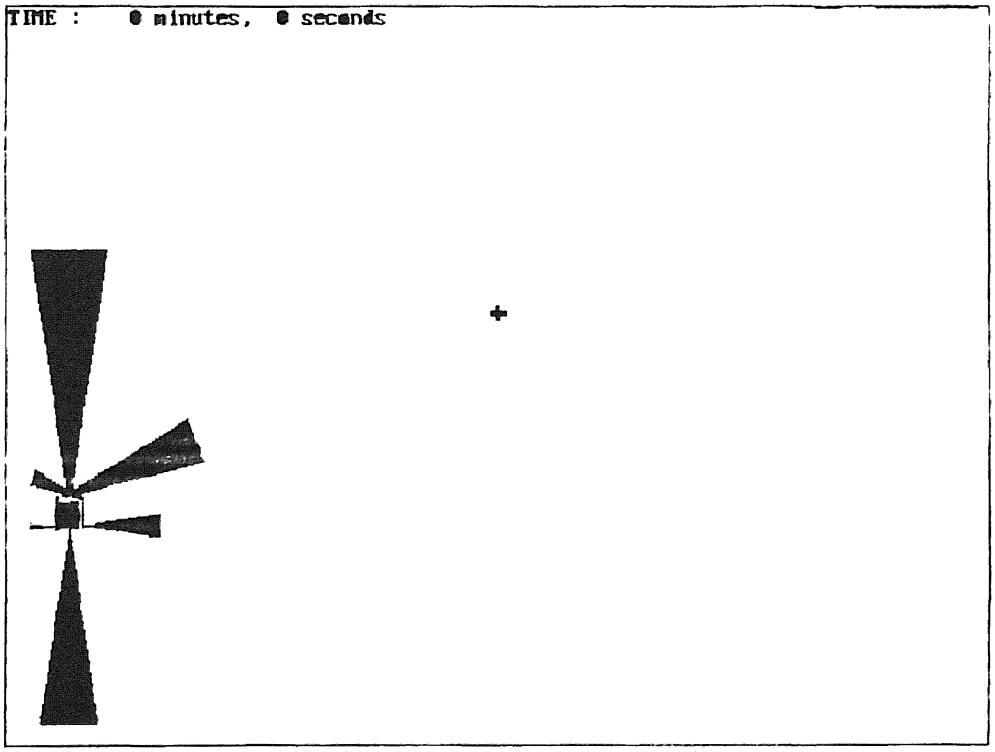

Figure 18. Example #1, Target #1 - Time = 0 Minutes, 0 Seconds.

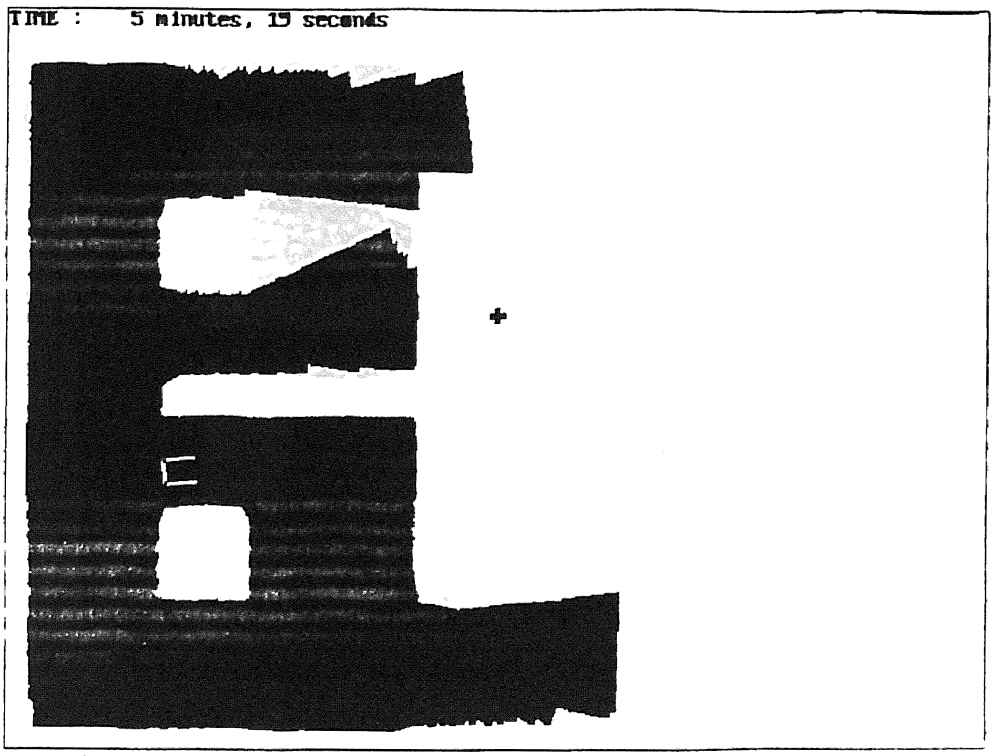

Figure 19. Example #1, Target #1 - Time = 5 Minutes, 19 Seconds.

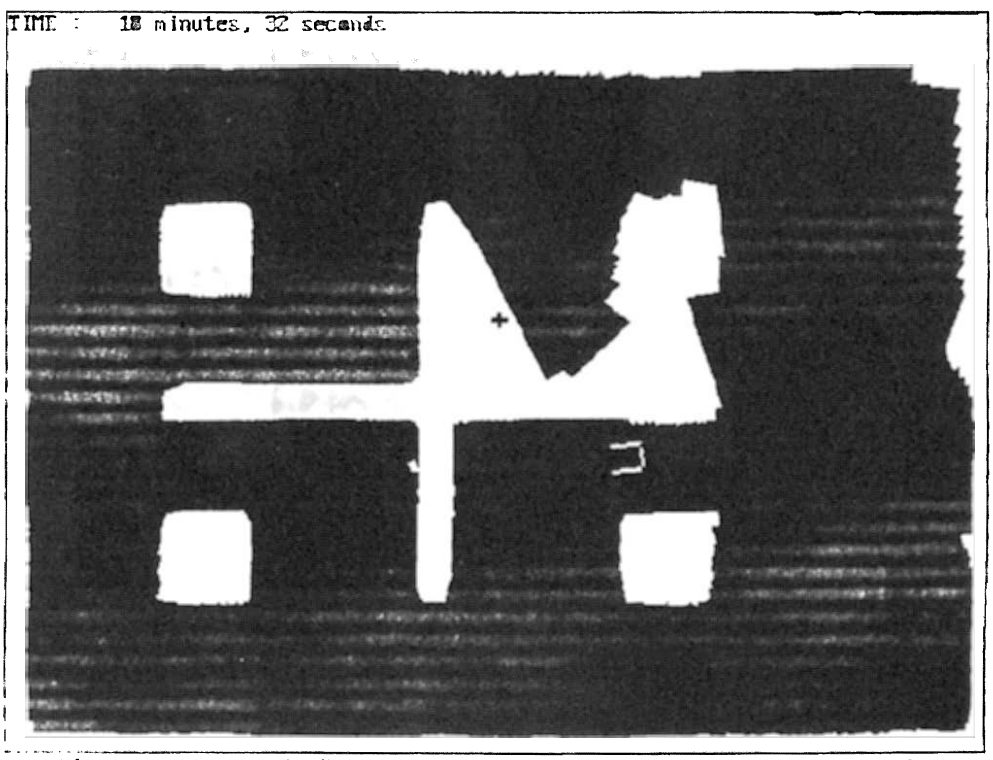

Figure 20. Example #1, Target #1 - Time = 10 Minutes, 32 Seconds.

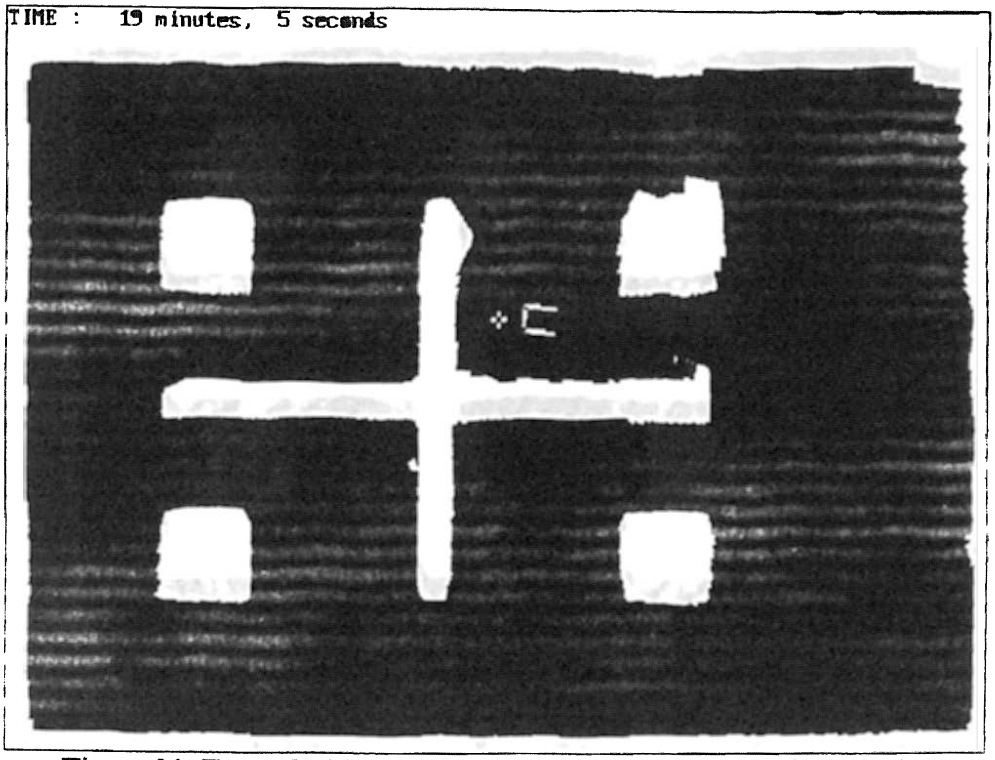

Figure 21. Example #1, Target #1 - Time = 19 Minutes, 5 Seconds.

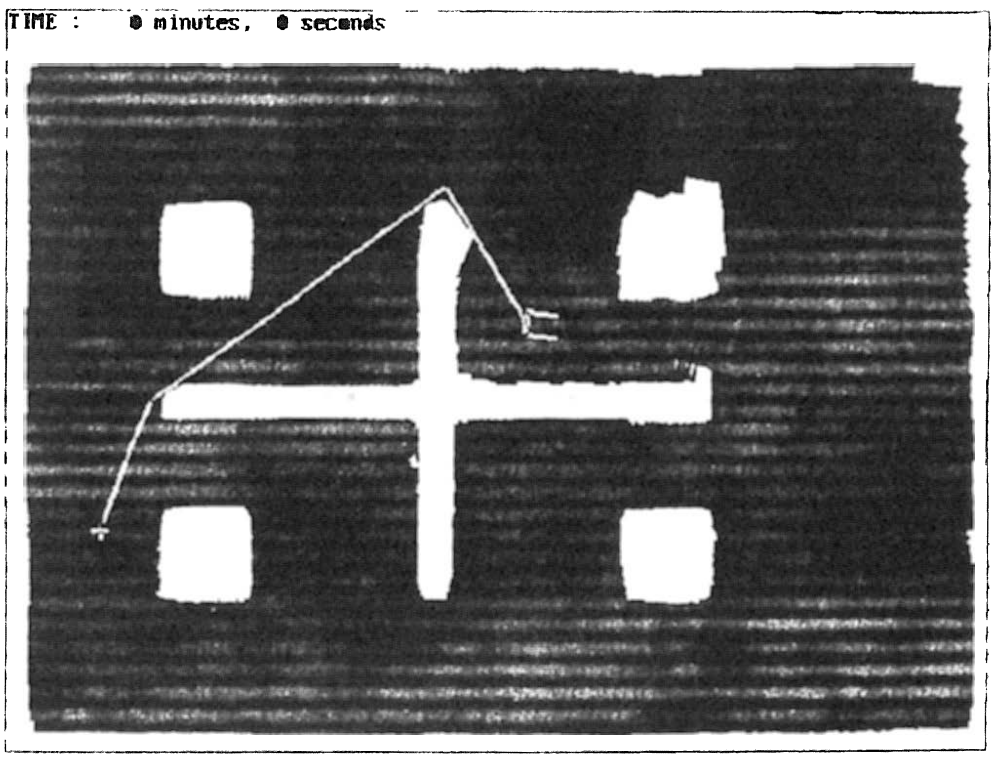

Figure 22. Example #1, Target #2 - Time = 0 Minutes, 0 Seconds.

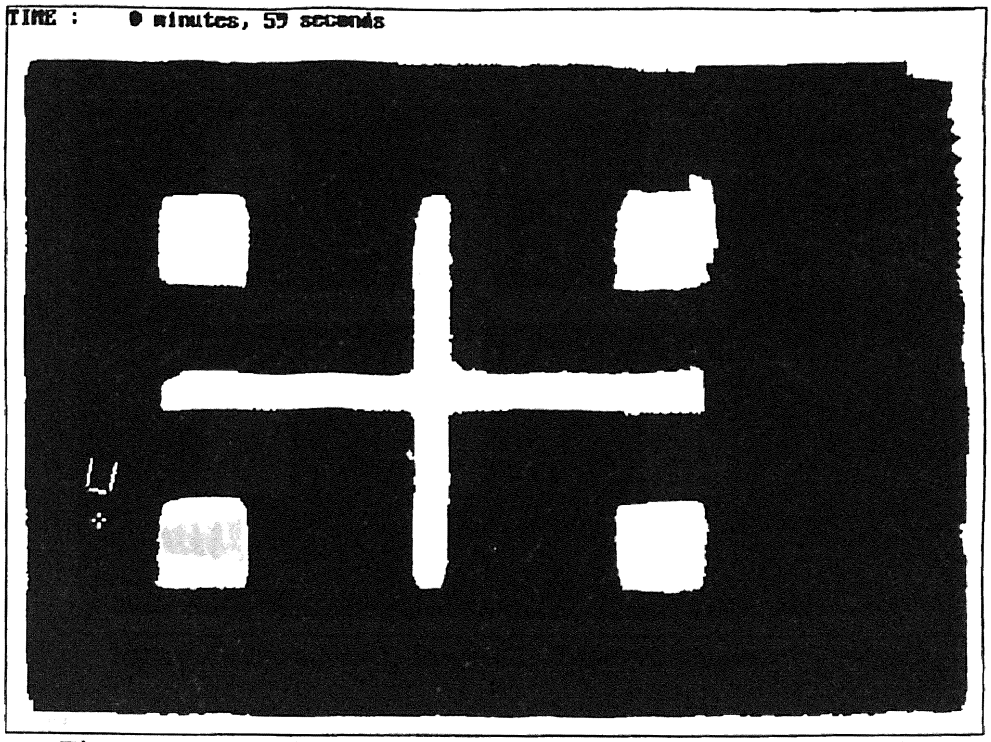

Figure 23. Example #1, Target #2 - Time = 0 Minutes, 59 Seconds.

Simulation Example # 2

Here the robot must avoid colliding with 10 convex obstacles found in the environment. As expected, the robot performed very well with only the sensor based path planner, although the mapped path planner performed very well also. The travel time for the sensor based path planner and the mapped path planner were 2 minutes, 13 seconds and 3 minutes and 1 second, respectively. The reason for such an improved performance for the sensor based path planner was the fact that it does not require many fuzzy rules to implement. Concave obstacle avoidance requires more complex rules to function properly. The mapped path planner required more time to reach the target due to conflicts with the collision avoidance algorithm. The first designated waypoint was reached after the robot encircled the closest obstacle, instead of reaching it in reverse, thus increasing the travel time. Figures 24 through 30 show the time history of both planning algorithms.

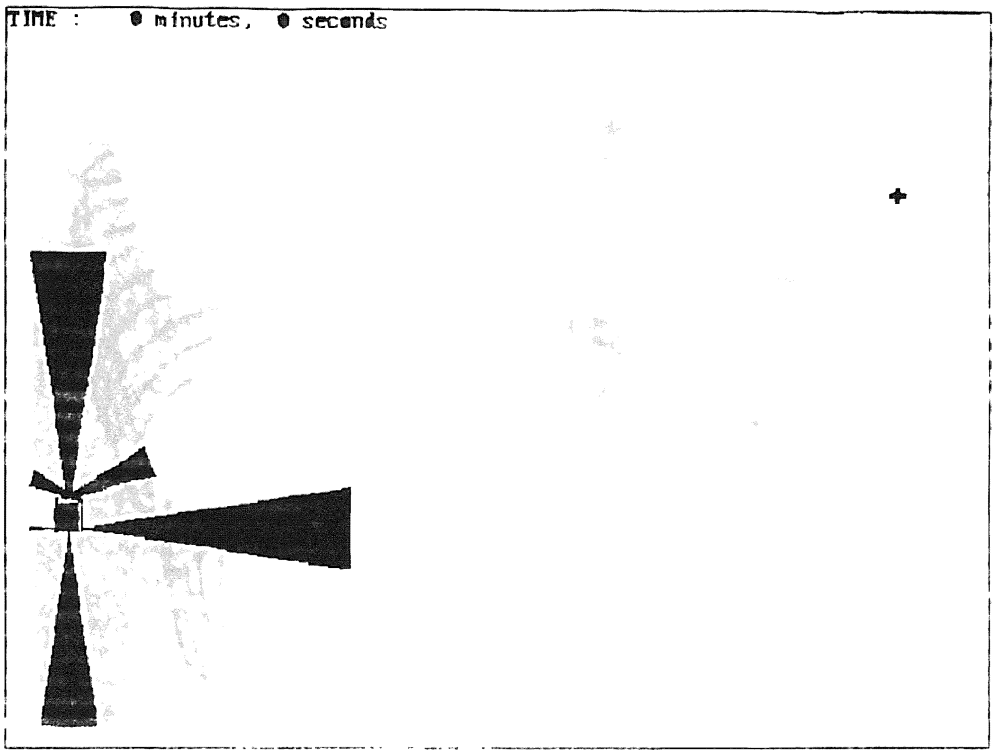

Figure 24. Example #2, Target #1 - Time = 0 Minutes, 0 Seconds.

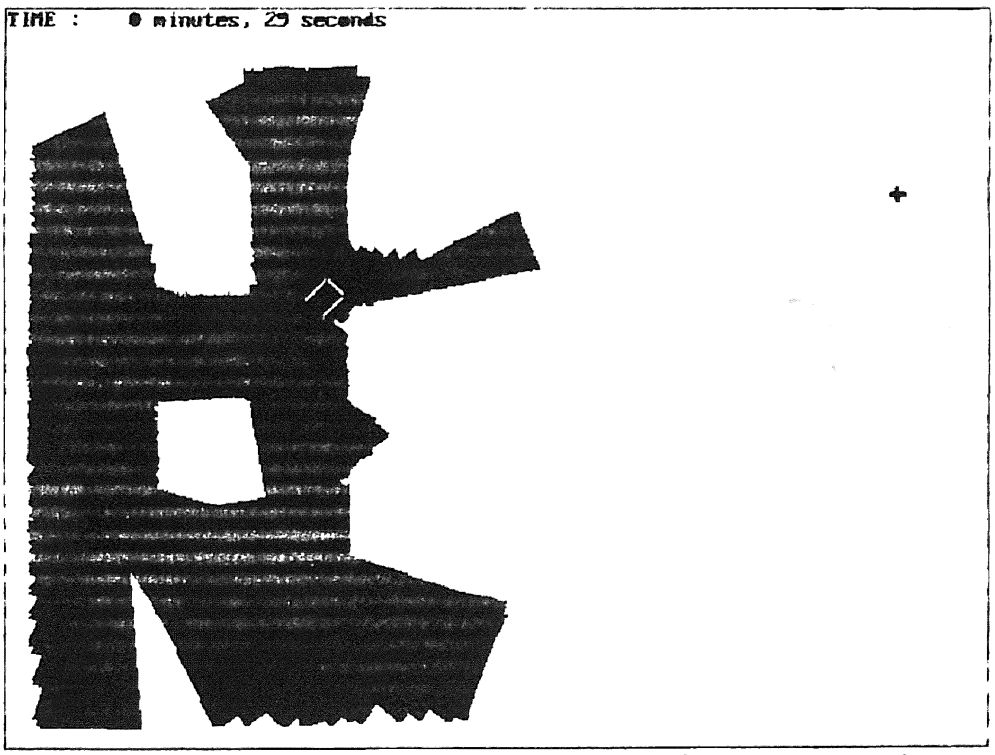

Figure 25. Example #2, Target #1 - Time = 0 Minutes, 29 Seconds.

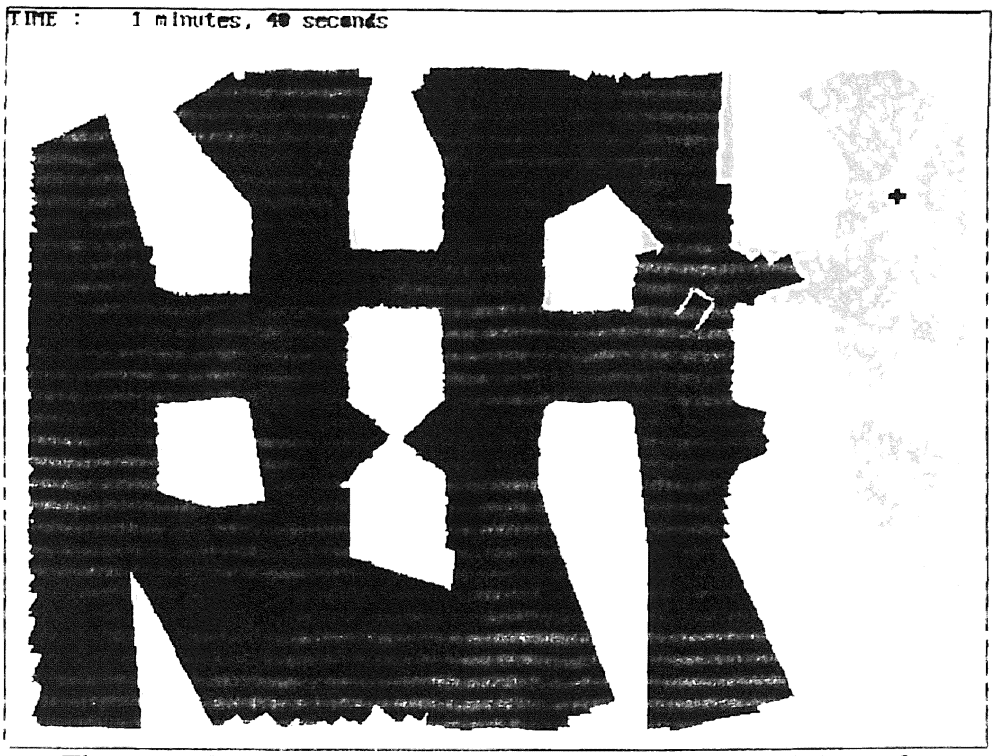

Figure 26. Example #2, Target #1 - Time = 1 Minutes, 40 Seconds.

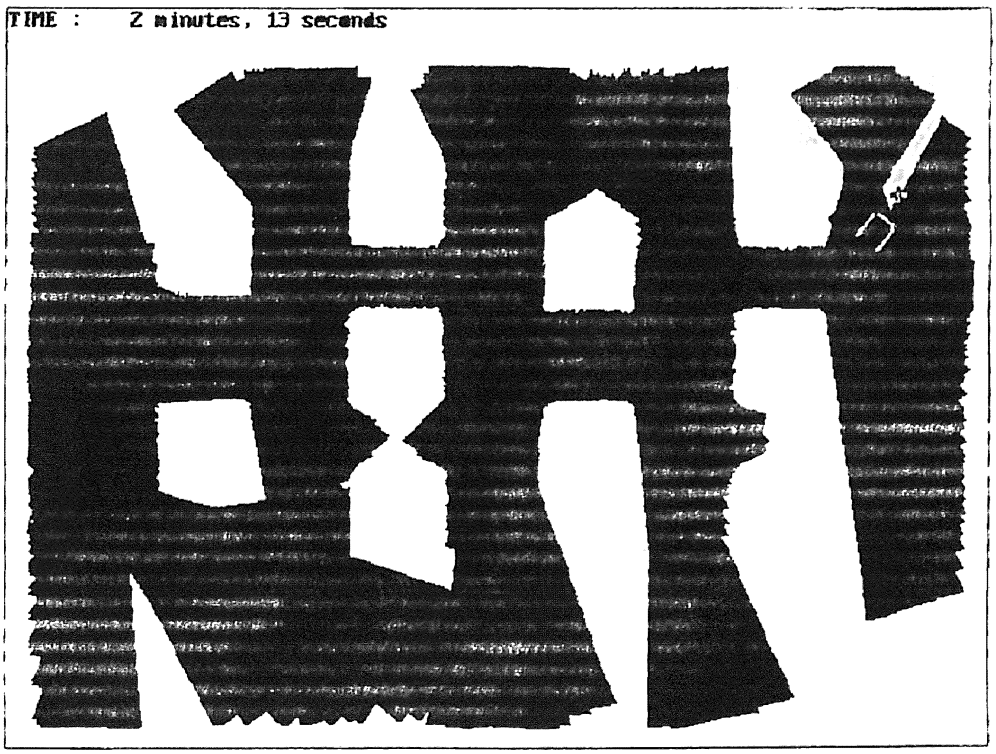

Figure 27. Example #2, Target #1 - Time = 2 Minutes, 13 Seconds.

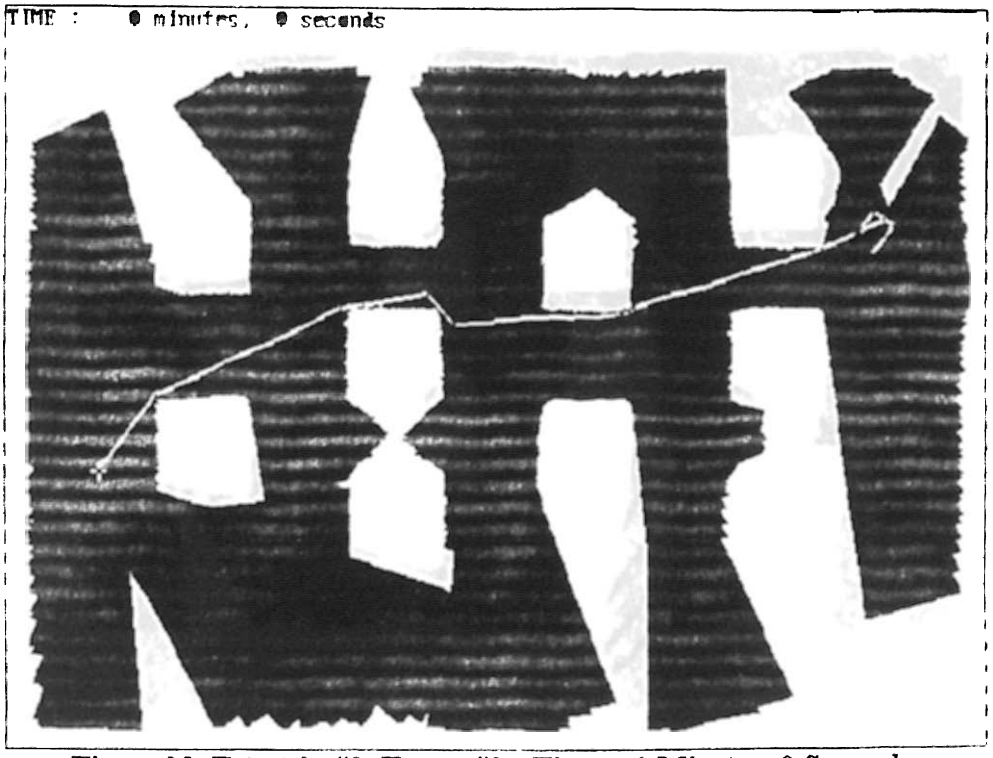

Figure 28. Example #2, Target #2 - Time = 0 Minutes, 0 Seconds.

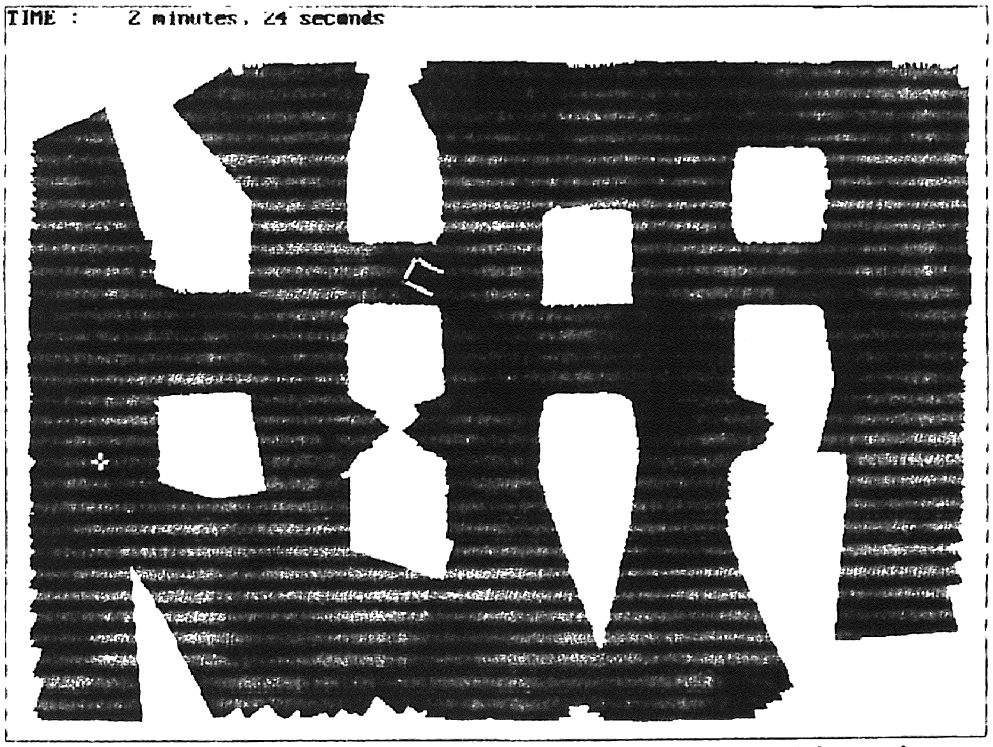

Figure 29. Example #2, Target #2 - Time = 2 Minutes, 24 Seconds.

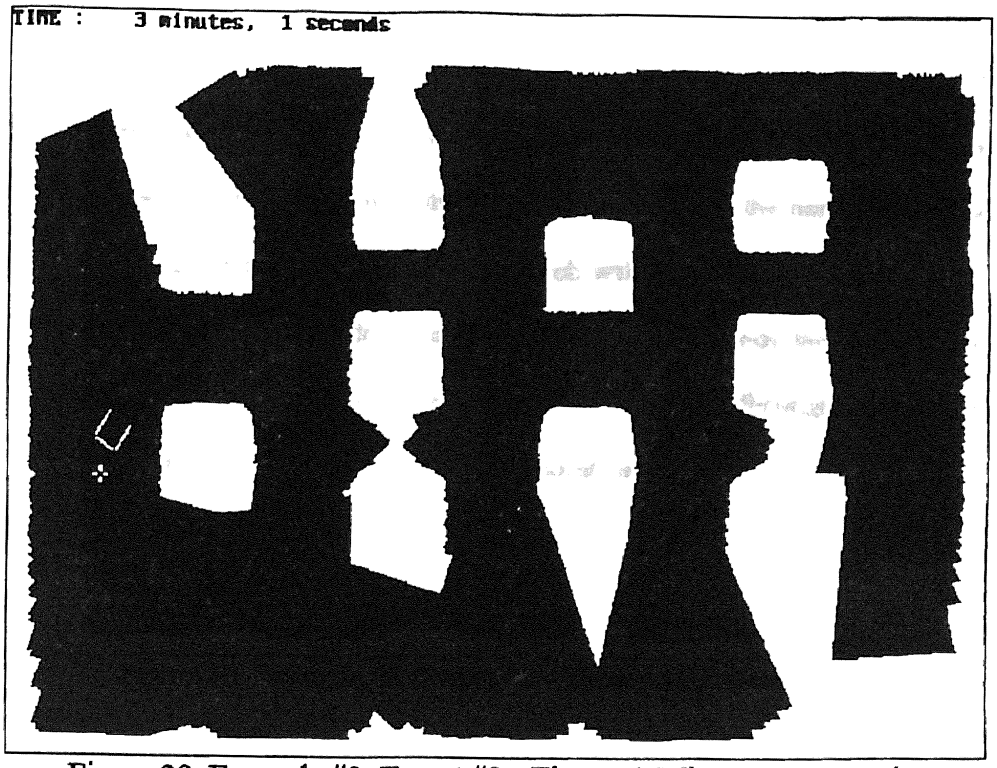

Figure 30. Example #2, Target #2 - Time = 3 Minutes, 1 Seconds.

Simulation Example #3

Here we have a simulated maze type environment. This simulation is a complete test of the capabilities of the robot. The travel time for the sensor based planner and the mapped path planner was 4 minutes, 46 seconds and 4 minutes, 12 seconds, respectively. Although the sensor based path planner functioned properly during this simulation, there were some instances in which the robot did not reach the designated target. Such situations arose only when using the sensor based path planner. Again, the reason behind the problem are logic insufficiencies behind the sensor based path planner algorithm. The improved version currently being developed should alleviate the problem.

The mapped path planner worked very efficiently, requiring only 4 minutes and 12 seconds to generate the required path. The generated path was sub-optimal, due to an incomplete map. The robot followed the designated path without any problems. There are

instances where the robot does not follow the generated path due to conflicts with the collision avoidance algorithm, particularly when traversing through narrow openings such as doors. The reason is that sometimes the robot is approaching the narrow opening at a fast speed, triggering the collision avoidance which will make the robot deviate from its original path. Modification of the fuzzy inference rules will remedy the problem and will be implemented in the next version of the controller. Figures 31 through 38, on pages 38 through 42, show the time history of the robot simulations.

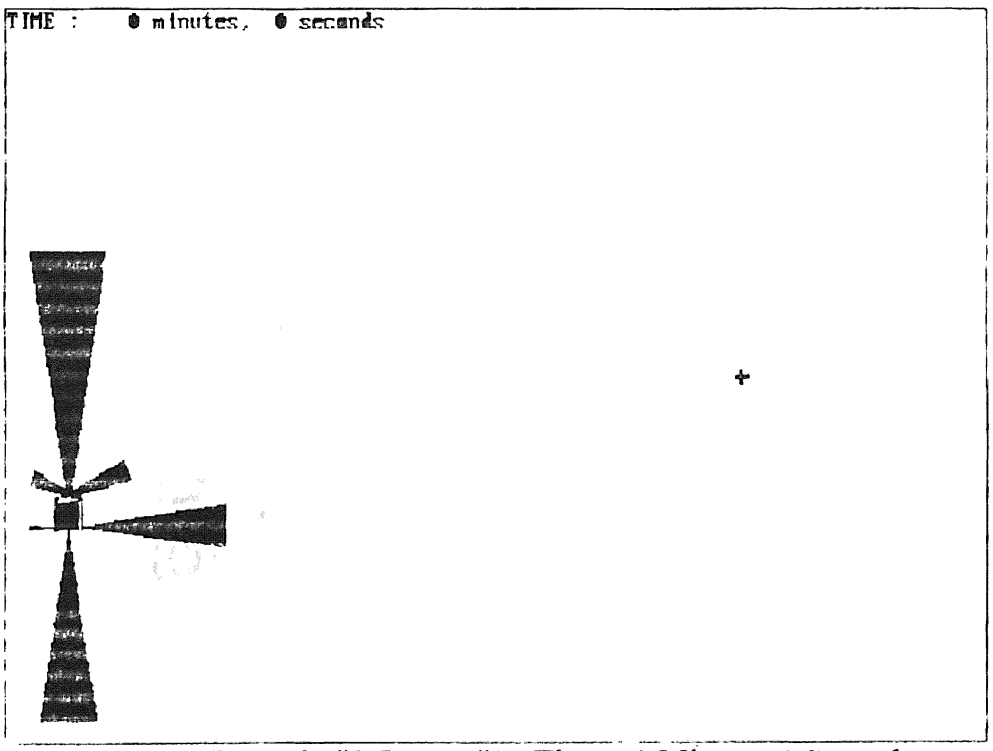

Figure 31. Example #3, Target #1 - Time = 0 Minutes, 0 Seconds.

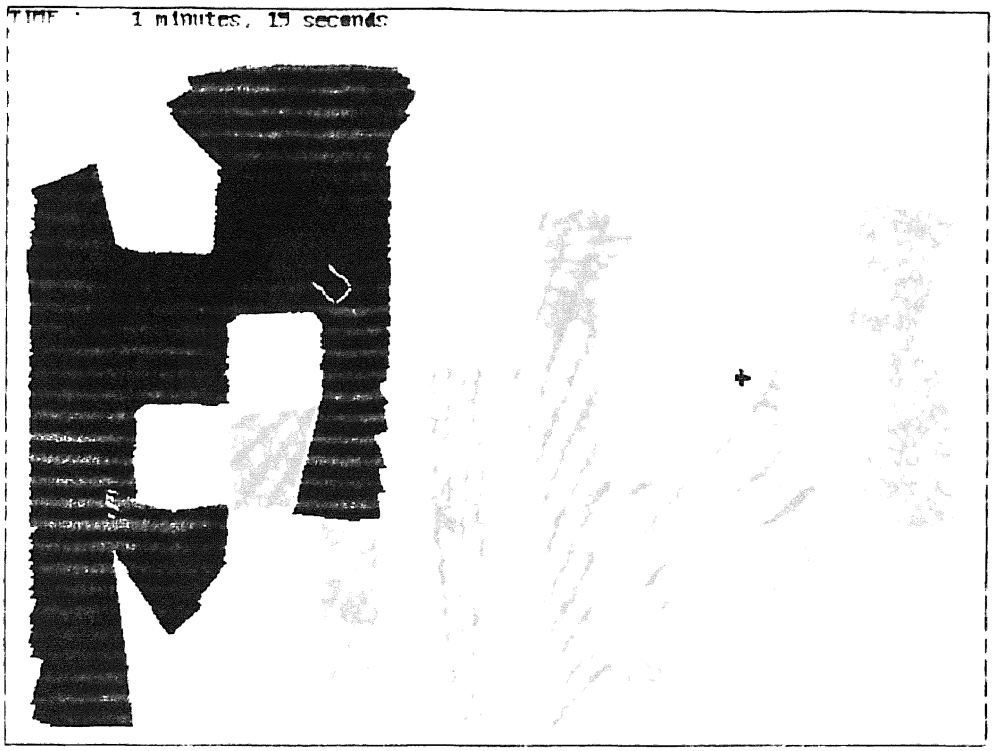

Figure 32. Example #3, Target #1 - Time = 1 Minutes, 19 Seconds.

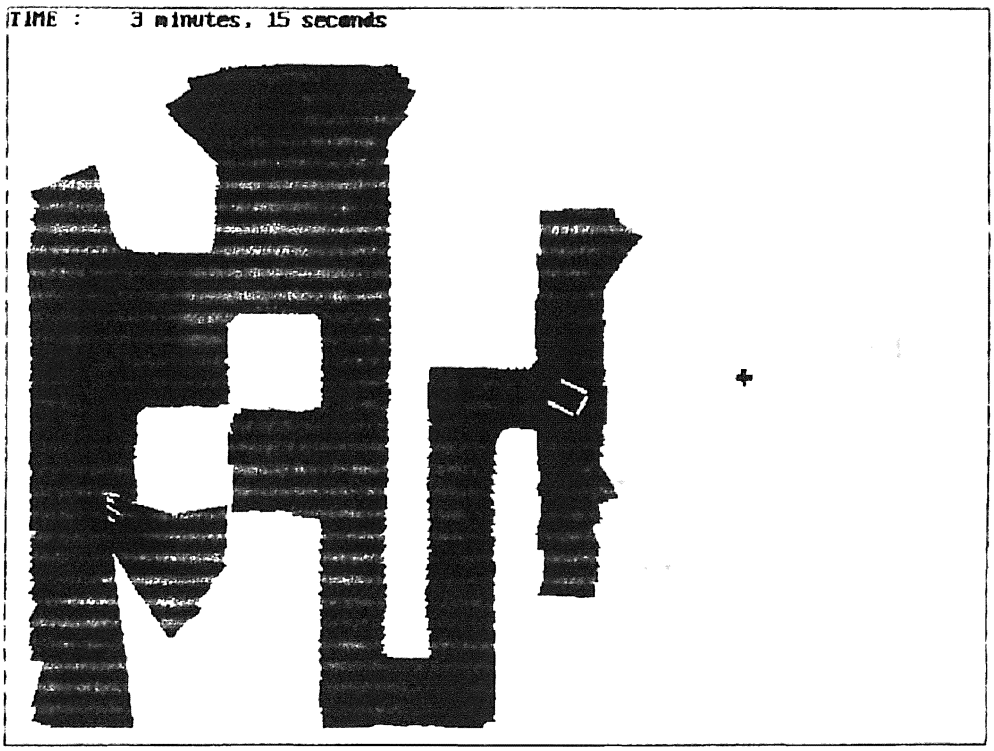

Figure 33. Example #3, Target #1 - Time = 3 Minutes, 15 Seconds.

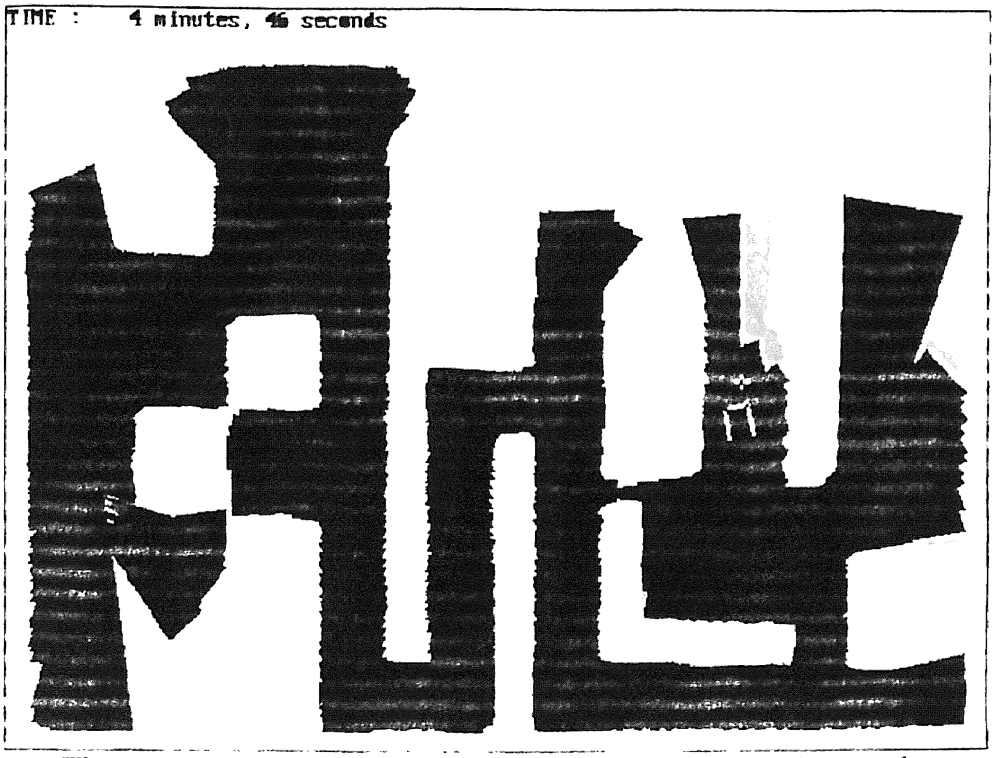

Figure 34. Example #3, Target #1 - Time = 4 Minutes, 46 Seconds.

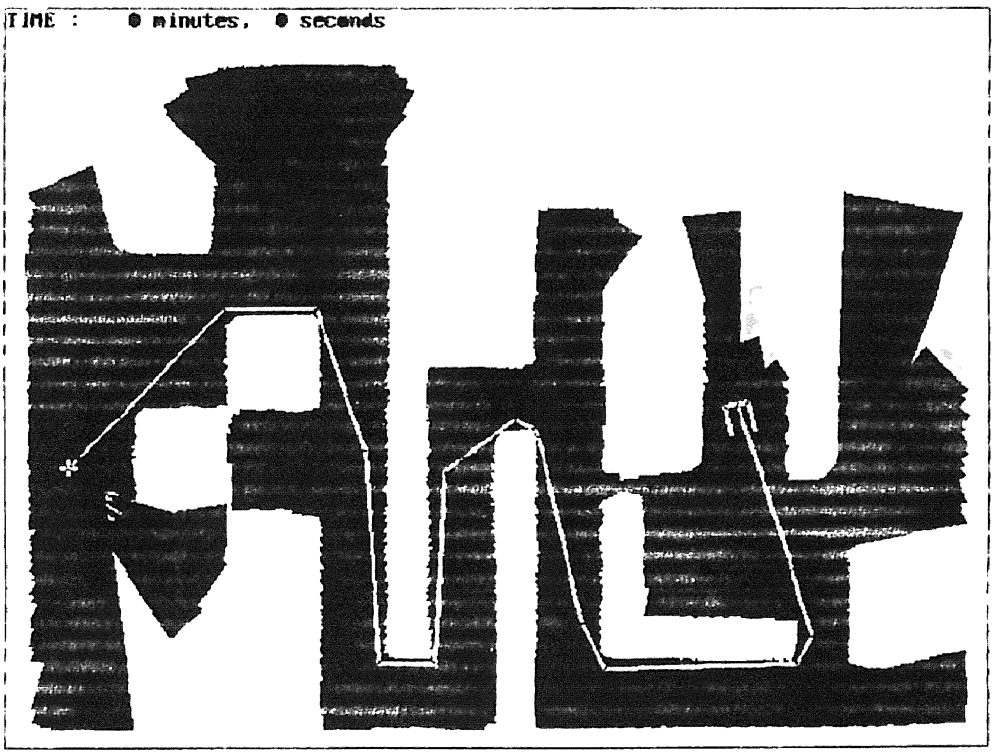

Figure 35. Example #3, Target #2 - Time = 0 Minutes, 0 Seconds.

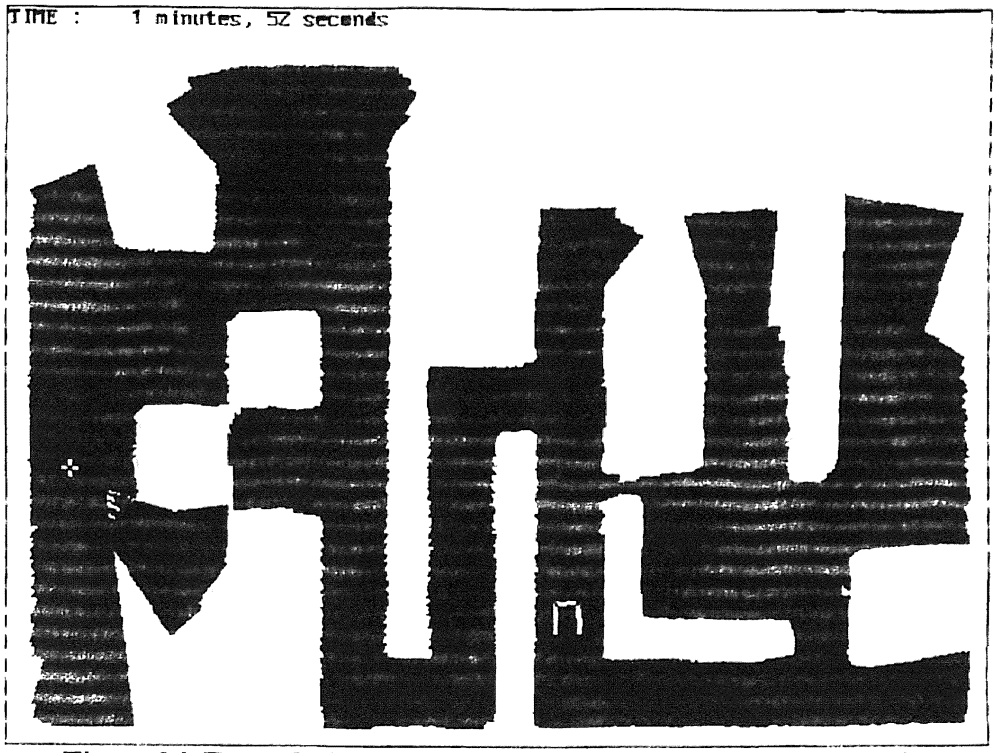

Figure 36. Example #3, Target #2 - Time = 1 Minutes, 52 Seconds.

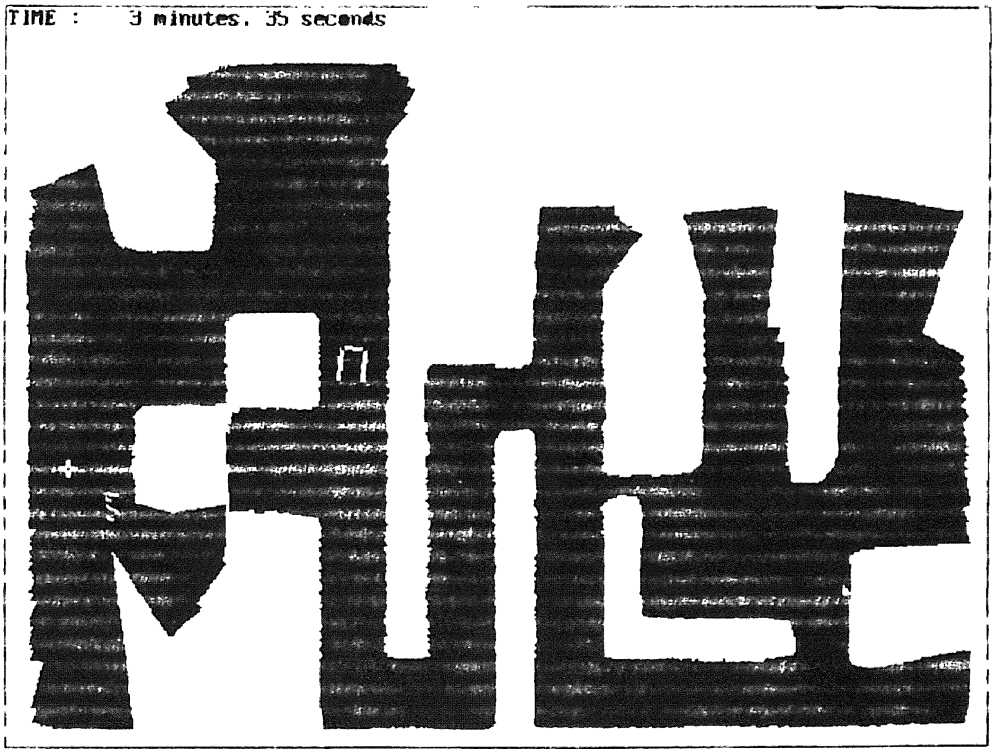

Figure 37. Example #3, Target #2 - Time = 3 Minutes, 35 Seconds.

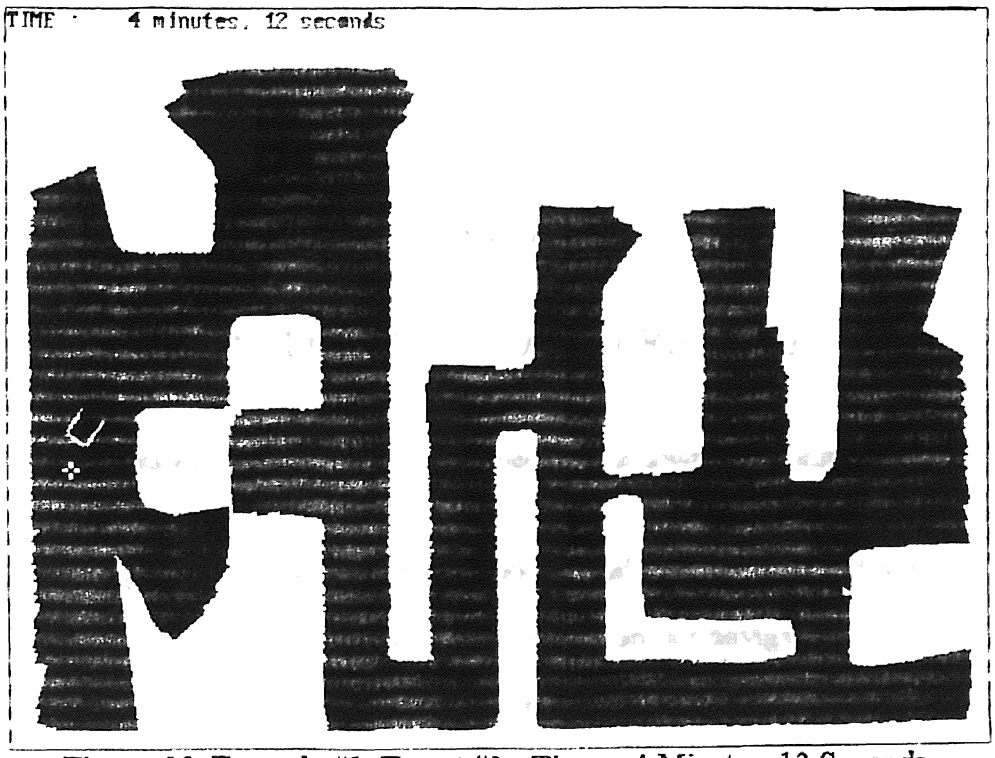

Figure 38. Example #3, Target #2 - Time = 4 Minutes, 12 Seconds.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

Autonomous Vehicle Performance in Unknown Environments

Fuzzy logic in combination with the sub-optimal path planner, and adaptive environment mapping provide a very effective mechanism for navigating and learning in unknown environments. The supervisor controller was designed to function in a controller area network environment. Immediate response for collision avoidance was addressed by considering a simple collision algorithm implemented in the vision module. This is in addition to the fuzzy logic collision avoidance subsystem of the supervisory controller, providing system robustness to component failure. High decisional and planning capabilities are implemented while maintaining fast response using a combination of fuzzy logic, adaptive environment mapping, and a very simple sub-optimal path planner. Sonar sensor range information is provided through a controller area network environment five times per second. The adaptive environment mapping algorithm is tailored to be used with sonar sensor information, and is very fast during execution, requiring only 50 milliseconds to update.

The author's contributions in the area of mobile robots is,

- A fast sub-optimal path planner based on an environment map.
- Fast adaptive mapping
- Complex planning capabilities while maintaining high reactivity using fuzzy logic in conjunction with the path planner.

All of the simulations were executed simultaneously on an INTEL 80486DX-33MHz based PC. The results obtained from the simulation examples of the autonomous robot show the supervisor module operated very quickly and efficiently using the mapped path planner algorithm, whereas the sensor based path planner was inadequate in the second simulation example. Problems that may be encountered when using the sensor based path planner is usually that the robot will get stuck in areas where there are concave obstacles. The time required to generate an average path was 0.6 seconds while the supervisor was able to go through one loop every 0.2 seconds. The robot arrived at every assigned location while avoiding various obstacles, and in the last experiment was able to go through narrow passages. The mapped path planner worked without any problems except when traversing narrow openings. Sometimes the robot would miss the entrance, although it would eventually turn around and go through it.

Future Work

When implemented each controller area network module will operate separately on slower computer boards, such as an INTEL 80386SX-16MHz based PC. The implementation of the robot will use the proposed supervisory controller connected in the CAN. Vision and propulsion modules are currently being implemented, including hardware, sensors, actuators, and software. New features such as optimized path planning for multiple targets will be devised and implemented. The mobile robot will be used as a test bed for developing a general design methodology for very complex distributed real-time control problems. Also, problems encountered during the simulation, particularly with the sensor base path planner, will be eliminated with an updated version of the sensor based path planner.

REFERENCES

1. The Truck & Bus Control and Communications Network Subcommittee of the Truck & Bus Electrical & Electronics Committee, Society of Agricultural Engineers (SAE). Recommended Practice for Serial Control and Communications Network (Class C) for Truck and Bus Applications. In SAE J1939, SAE Publications, 1993.

2. Fabrice R. Noreils and Roland Prajoux. From Planning to Execution Monitoring Control for Indoor Mobile Robots. In Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California - April 1991.

3. Hubert A. Vasseur, Francois G. Pin, and Jack R. Taylor. Navigation of a Car-like Mobile Robot using a Decomposition of the Environment in Convex Cells. In Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California - April 1991.

4. Alexander Zelinsky. A Mobile Robot Exploration Algorithm. IEEE Transactions on Robotics and Automation, Vol. 8, No. 6, December 1992

5. Alois A. Holenstein amd Essam Badreddin. Collision Avoidance in a Behavior-based Mobile Robot Design. In Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California - April 1991.

6. von Altrock, B. Krause and H.-J. Zimmermann. Advanced fuzzy logic control of a model car in extreme situations. Fuzzy Sets and Systems 48 , pp. 41-52, 1992.

7. Khatib. Real-time Obstacle Avoidance for Manipulators and Mobile Robots. IEEE Conference on Robotics and Automation, pp. 500-505, 1985

8. Krogh and C, Thorpe. Integrated Path Planner and Dynamic Steering Control for Autonomous Vehicles. IEEE Conference on Robotics and Automation, pp. 1664-1669, 1986

9. Martin Beckerman and E. M. Oblow. Treatment of Systematic Errors in the Processing of Wide-Angle Sonar Sensor Data for Robotic Navigation. Transactions on Robotics and Automation, Vol. 6, No. 2, April 1990.

10. Han-Pang Huang and Pei-Chien Lee. Microprocessor-Based Control of Autonomous Mobile Robots with Obstacle Avoidance. Proceedings of the 30th Conference on Decision and Control, Brighton, England December 1991.

11. H.-J. Zimmerman, Fuzzy Set Theory - and its Applications, 2nd rev. edn. (Kluwer, Dordrecht - Boston, 1991).

12. Andersen, C.S., Madsen, C.B., Sorensen, J.J., Kirkeby, N.O.S., Jones, J.P. and Christensen, H.I., Navigation using range images on a mobile robot. Robotics and Autonomous Systems, October 1992 pp. 147 - 160.

APPENDIXES

APPENDIX A

SIMULATION MODEL ASSUMPTIONS

Robot Dynamic Assumptions

The kinematic motion of the robot is controlled by the velocity of the front wheels and the steering angle of the front wheels. Figure 39 shows the geometric assumptions of the mobile robot.

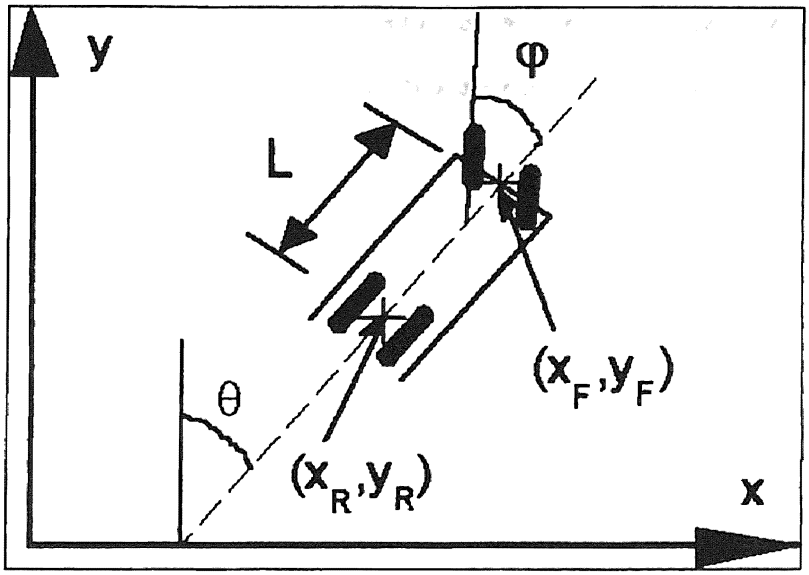

Figure 39. Kinematic Constraints for Car-Like Mobile Robot.

From [12] we have

$$\dot{x}_F = v \cdot \cos(\theta + \varphi), \quad \dot{y}_F = v \cdot \sin(\theta + \varphi), \quad \dot{\theta} = \frac{v \cdot \sin(\varphi)}{L}. \tag{2}$$

a discretized version of eq. (2) is used for the robot simulation. The dc-motor dynamics consist of a second order system while the steering mechanism is assumed to have a first order response. The dc-motor's inertia used reflects a nominal size including the total weight of the robot. Tire slip and rolling resistance is assumed negligible since the robot will be operating at low speeds.

Sonar Dynamics Assumptions

The sonar dynamics are assumed to be instantaneous since we are sampling at 5Hz and it is possible to obtain sonar information at 8.25 Hz assuming a worst case scenario of 10 meter range for all the sonar. Since we will be triggering the sonar in two stages, the total distance travelled is 40 meters for each sample. The speed of sound, 330 m/s, divided by 40 m equals 8.25 Hz. In the final implementation, the maximum range that will be allowed for the sonar will be less than 10 m, thus the maximum allowable sampling rate will also increase.

APPENDIX B

SIMULATION SOFTWARE CODE

```
/*****************************************************************************
 FILE            : ZZ_CAN.H

 DESCRIPTION  : CONTROLLER AREA NETWORK FUNCTIONS
                  FOR AUTONOMOUS VEHICLE

                  This header file contains code for CAN COMMUNICATIONS
-----------------------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE : APRIL 27, 1993
*****************************************************************************/
#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));
```

```
/*************************************************************************
 FILE            : ZZ_CAN.C

 DESCRIPTION     : CONTROLLER AREA NETWORK FUNCTIONS
                   FOR AUTONOMOUS VEHICLE

                   This file contains code for CAN COMMUNICATIONS
 ----------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE     : APRIL 27, 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);
}
```

```
/*****************************************************************************
 FILE            : ZZ_CARSP.H

 DESCRIPTION  : MOBILE ROBOT SPECIFIC PARAMETERSFUNCTIONS

-----------------------------------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE  : APRIL 27, 1993
*****************************************************************************/


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.H

 DESCRIPTION  : CONSOLE MODULE FOR AUTONOMOUS VEHICLE

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

 by              : Ricardo Andujar

 LAST UPDATE  : APRIL 27, 1993
*****************************************************************************/
void    keypress_handler(void);

void    kbsig(void);
```

```
/*****************************************************************************
 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] = getcbuf(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 'T': *NewTarget = FIRST_TIME;
/*                      case 8:         backspace_handler();
                        break;
                        case 13: return_handler();
                        break;
                        default:    putcbuf(cc,keybuf2);
                                ungetcbuf(cc,keybuf);
                                printchar_handler();
                        break;
*/
                        }
        }
}
```

```
/*************************************************************************
 FILE           : ZZ_FUZZY.H

 DESCRIPTION    : FUZZY LIBRARY HEADER
                  This header file contains code for fuzzy Inference library
 ------------------------------------------------------------------------

 by             : Ricardo Andujar
                  Fuzzy Inference Library
                  Copyright 1993

 LAST UPDATE    : APRIL 27, 1993
*************************************************************************/

/*************************************************************************
********
********                  MEMBERSHIP TYPE DEFINITIONS
********
*************************************************************************/
typedef struct
        {
                double  bottomleft,
                        topleft,
                        topright,
                        bottomright;
        } ZZ_TRAPEZOID;

typedef struct
        {
                int     discretenum;
                double  lowrange,
                        highrange,
                    *discrete;
        } ZZ_FUZZYOUTPUT;

/*************************************************************************
********
********    FUZZY LIBRARY FUNCTIONS FOR ALL TYPE OF INPUTS
********
*************************************************************************/
void    ZZ_InitFuzzyOutput(double lowrange,double highrange,
                        ZZ_FUZZYOUTPUT *,int discretenum);
void    ZZ_DelFuzzyOutput(ZZ_FUZZYOUTPUT *);
void    ZZ_AddMax(double min,ZZ_TRAPEZOID *,ZZ_FUZZYOUTPUT *);
double  ZZ_Defuzzify(ZZ_FUZZYOUTPUT *);
void    ZZ_ClearFuzzyOutput(ZZ_FUZZYOUTPUT *);
/*************************************************************************
********
********    FUZZY LIBRARY FUNCTIONS FOR SINGLETON INPUTS
********
*************************************************************************/
double  ZZ_Member(double,ZZ_TRAPEZOID *);
double  ZZ_Max(double,double);
double  ZZ_Min(double,double);
double  ZZ_Complement(double,ZZ_TRAPEZOID *);
```

```
/*****************************************************************************
 FILE            : ZZ_FUZZY.C

 DESCRIPTION  : FUZZY LIBRARY FILE FOR AUTONOMOUS VEHICLE

                This file contains code for the fuzzy logic library.
-----------------------------------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE  : APRIL 27, 1993
*****************************************************************************/


/*****************************************************************************
 INCLUDE FILES FOR FUZZY LIBRARY
 *****************************************************************************/
#include"ZZ_fuzzy.h"
#include<alloc.h>

/*****************************************************************************
 *
 *  Membership Function Value for Singleton Input
 *  Applies only to Trapezoidal Membership Functions          *
 *                                                            *
 *****************************************************************************/
double   ZZ_Member(double  value,ZZ_TRAPEZOID *shape)
{

/*1*/    if(shape->topleft > value && shape->bottomleft <value)
                return((value-shape->bottomleft)/
                        (shape->topleft-shape->bottomleft));

/*2*/    if(shape->topright < value && shape->bottomright >value)
                return(-(value-shape->bottomright)/
                (shape->bottomright-shape->topright));

/*3*/    if(shape->topleft >= shape->bottomleft && shape->bottomleft>=value ||
        shape->topright <= shape->bottomright && shape->bottomright<=value)
                return(0.0);

/*4*/    return(1.0);

}


/*****************************************************************************
 *
 *         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);
}


/*****************************************************************************
 *
 *  Complement of Membership Function Value for Singleton Input
 *  Applies only to Trapezoidal Membership Functions
 *
 *****************************************************************************/
double   ZZ_Complement(double  value,ZZ_TRAPEZOID *shape)
{
/*1*/   if(shape->topleft > value && shape->bottomleft <value)
                return(1-(value-shape->bottomleft)/
                        (shape->topleft-shape->bottomleft));

/*2*/   if(shape->topright < value && shape->bottomright >value)
                return(1+(value-shape->bottomright)/
                (shape->bottomright-shape->topright));

/*3*/   if(shape->topleft >= shape->bottomleft && shape->bottomleft>=value ||
        shape->topright <= shape->bottomright && shape->bottomright<=value)
                return(1.0);

/*4*/   return(0.0);

}


/*****************************************************************************
 *
 *  ALLOCATES MEMORY FOR MEMBERSHIP VALUES BETWEEN A GIVEN RANGE
 *
 *****************************************************************************/
void    ZZ_InitFuzzyOutput(double lowrange,double highrange,
                        ZZ_FUZZYOUTPUT *funct,int discretenum)
{
        funct->lowrange = lowrange;
        funct->highrange = highrange;
        funct->discretenum = discretenum;
        funct->discrete = farmalloc(8*(discretenum+1));
}
```

```
/*****************************************************************************
 *
 *  DEALLOCATES MEMORY FOR MEMBERSHIP VALUES BETWEEN A GIVEN RANGE
 *
 *****************************************************************************/
void    ZZ_DelFuzzyOutput(ZZ_FUZZYOUTPUT *funct)
{
        farfree(funct->discrete);
}


/*****************************************************************************
 *
 *      COMPARES TRAPEZOIDAL MEMBERSHIP FUNCTION WITH MEMBERSHIP
 *      VALUES STORED IN DISCRETE RANGE
 * ---------------------------------------------------------------------------
 *      LARSEN'S RULE IS USED WHEN COMPARING THE MEMBERSHIP VALUES
 *
 *****************************************************************************/
void    ZZ_AddMax(double min,ZZ_TRAPEZOID *memb,ZZ_FUZZYOUTPUT *funct)
{
        double dx;
        int i;

        dx = (funct->highrange- funct->lowrange)/
                ((double)(funct->discretenum-1));

        for(i=0;i<funct->discretenum;i++)
        {
        funct->discrete[i] = ZZ_Max(min*ZZ_Member(funct->lowrange+dx*(double)i,
                                                memb),funct->discrete[i]);
        }
}


/*****************************************************************************
*
*       DEFUZZIFICATION USING THE MEAN VALUE
*
*****************************************************************************/
double  ZZ_Defuzzify(ZZ_FUZZYOUTPUT *funct)
{
        double dx,den=0,num=0;
        int i;

        dx = (funct->highrange-funct->lowrange)/
                ((double)(funct->discretenum-1));
        for(i=0;i<funct->discretenum;i++)
        {
                num += (funct->lowrange+dx*(double)i)*funct->discrete[i];
                den += funct->discrete[i];
        }
        if(den==0)
                return(0.0);
        return(num/den);
```

```
}

/*************************************************************************
 *                                                                       *
 *        RESET ALL DISCRETE RANGE MEMBERSHIP VALUES TO ZERO             *
 *                                                                       *
 *************************************************************************/
void    ZZ_ClearFuzzyOutput(ZZ_FUZZYOUTPUT *funct)
{
        int i;

        for(i=0;i<funct->discretenum;i++)
                funct->discrete[i] = 0.0;
}
```

```
/*****************************************************************************
FILE            : ZZ_GRAPH.H

DESCRIPTION     : SUPERVISOR MODULE FOR AUTONOMOUS VEHICLE

                  This file contains header file information for 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     : APRIL 27, 1993
*****************************************************************************/


void        ZZ_InitGraph(void);
void        ZZ_CloseGraph(void);
void        ZZ_DrawBlackBox(void);
void        ZZ_Polar2Cart(double ang,double r,double *x,double *y);
void        ZZ_Cart2Polar(double x,double y,double *ang,double *r);
void        ZZ_Real2Screen(double x,double y,int *xc, int *yc);
void        ZZ_Screen2Real(int xc,int yc,double *x, double *y);
```

```
/*****************************************************************************
 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  : APRIL 27, 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  : APRIL 27, 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 impleneted, each module will have it's own
 **********      separate loop on different processors.
 *****************************************************************************/
```

```
        while(1)
        {
        /*********        Propulsion Loop
         *********        ---------------
         *********        For every Supervisor Loop, the Propulsion
         *********        Module Loops Four Times with 0.05 sample period
         *********        for the actuator controls.
         *********/
                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_MAP.H

DESCRIPTION  : ADAPTIVE MAPPING FILE FOR AUTONOMOUS VEHICLE

                This file contains header file 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  : APRIL 27, 1993
*****************************************************************************/
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);
```

```
/*************************************************************************
 FILE            : ZZ_MAP.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 : APRIL 27, 1993
*************************************************************************/


/*************************************************************************
 INCLUDE FILES FOR MAPPING FILE : SUPERVISOR MODULE
 *************************************************************************/
#include<graphics.h>
#include<time.h>
#include<stdlib.h>
#include<math.h>
#include"ZZ_CAN.H"
#include"ZZ_SUPR2.H"
#include"ZZ_CARSP.H"
#include"ZZ_MAP.H"

extern  int
        graph;

static  double
        *CarXc,
        *CarYc,
        *Xc,        /**** X SENSOR POSITIONS (pixels) ****/
        *Yc,        /**** Y SENSOR POSITIONS (pixels) ****/
        *Range,
        *Compass,
        *RoadSpeed;


static int
        *Xconvert,
        *Yconvert,
        Crash;
/*************************************************************************
 *
 *      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)
{
        CarXc = one;
        CarYc = two;
        Xc = three;
        Yc = four;
        Range = five;
        Compass = six;
        RoadSpeed = seven;
        Xconvert = eight;
        Yconvert = nine;
}


/*************************************************************************
 *
 *      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)
{

        ZZ_DrawRoom();
        ZZ_DrawGrayArea();
        ZZ_DrawBlackBox();
        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     i,j,radius,poly[12],color=BLACK,temp[6];
                float temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8;

        setfillstyle(SOLID_FILL,BLACK);
        for(i=0;i<NUM_SENSORS;i++)
        {
                int     xcenter=Xc[i]+*Xconvert,
                        ycenter=Yc[i]+*Yconvert,
                radius = (int)(Range[i]*SCREENCONVERT);
```

```
                poly[0] = xcenter+(int)((float)radius
                        *-sin(-*Compass-NominalAngle[i]+HALFSPREADANGLE*1.3));
                poly[1] = ycenter+(int)((float)radius
                        *-cos(-*Compass-NominalAngle[i]+HALFSPREADANGLE*1.3));
                poly[2] = xcenter+(int)((float)radius
                        *-sin(-*Compass-NominalAngle[i]-HALFSPREADANGLE*1.3));
                poly[3] = ycenter+(int)((float)radius
                        *-cos(-*Compass-NominalAngle[i]-HALFSPREADANGLE*1.3));

                poly[4] = xcenter+(int)((float)radius
                        *-sin(-*Compass-NominalAngle[i]));
                poly[5] = ycenter+(int)((float)radius
                        *-cos(-*Compass-NominalAngle[i]));

                poly[6] = xcenter+(int)((float)radius*1.3
                        *-sin(-*Compass-NominalAngle[i]+HALFSPREADANGLE*1.3));
                poly[7] = ycenter+(int)((float)radius*1.3
                        *-cos(-*Compass-NominalAngle[i]+HALFSPREADANGLE*1.3));
                poly[8] = xcenter+(int)((float)radius*1.3
                        *-sin(-*Compass-NominalAngle[i]-HALFSPREADANGLE*1.3));
                poly[9] = ycenter+(int)((float)radius*1.3
                        *-cos(-*Compass-NominalAngle[i]-HALFSPREADANGLE*1.3));

                poly[10] = Xc[i]+*Xconvert+(int)((float)radius*1.25
                        *-sin(-*Compass-NominalAngle[i]));
                poly[11] = Yc[i]+*Yconvert+(int)((float)radius*1.25
                        *-cos(-*Compass-NominalAngle[i]));

                temp1 = (poly[6]-poly[0])/15.;
                temp2 = (poly[7]-poly[1])/15.;
                temp3 = (poly[8]-poly[2])/15.;
                temp4 = (poly[9]-poly[3])/15.;
                temp5 = (poly[8]-poly[6])/15.;
                temp6 = (poly[9]-poly[7])/15.;
                temp7 = (poly[10]-poly[4])/15.;
                temp8 = (poly[11]-poly[5])/15.;
/*****************************************************************
 Check for Moving or New Objects in Area
 *****************************************************************/
                for(j=0;j<15;j++)
                {
                color=getpixel(poly[0]+j*temp1,poly[1]+j*temp2);
                    if(color==LIGHTGRAY || color==WHITE || color==LIGHTRED)
                                        break;
                color=getpixel(poly[2]+j*temp3,poly[3]+j*temp4);
                    if(color==LIGHTGRAY || color==WHITE || color==LIGHTRED)
                                        break;
                color=getpixel(poly[6]+j*temp5,poly[7]+j*temp6);
                    if(color==LIGHTGRAY || color==WHITE || color==LIGHTRED)
                                        break;
                color=getpixel(poly[4]+j*temp7,poly[5]+j*temp8);
                    if(color==LIGHTGRAY || color==WHITE || color==LIGHTRED)
                                        break;
                }
```

```
                switch(color)
                {
/*************************************************************************
 Place gray patch on moving/new obstacle
 *************************************************************************/
                default:
                if(radius>15 && radius<SMAXRANGE-2)
                {
                setwritemode(0);
                setcolor(BLACK);
                setfillstyle(SOLID_FILL,LIGHTGRAY);
                poly[4] = poly[8];
                poly[5] = poly[9];
                fillpoly(4,poly);
            }

/*************************************************************************
 Clear Free Space in Global Map
 *************************************************************************/
                case LIGHTGRAY:
                case WHITE:
                case LIGHTRED:
                if(radius>14)
                {
                setwritemode(1);
                setcolor(BLACK);
                setfillstyle(SOLID_FILL,BLACK);
                poly[0] = xcenter+(int)((float)radius
                        *-sin(-*Compass-NominalAngle[i]+HALFSPREADANGLE*.85));
                poly[1] = ycenter+(int)((float)radius
                        *-cos(-*Compass-NominalAngle[i]+HALFSPREADANGLE*.85));
                poly[2] = xcenter+(int)((float)radius
                        *-sin(-*Compass-NominalAngle[i]-HALFSPREADANGLE*.85));
                poly[3] = ycenter+(int)((float)radius
                        *-cos(-*Compass-NominalAngle[i]-HALFSPREADANGLE*.85));
                poly[4] = Xc[i]+*Xconvert;
                poly[5] = Yc[i]+*Yconvert;
                fillpoly(3,poly);
                }
                break;
                }

        }

}


/*************************************************************************
 *
 *      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 += getpixel(TX+res,TY+res);
        sum += getpixel(TX+res,TY-res);
        sum += getpixel(TX-res,TY+res);
        sum += getpixel(TX-res,TY-res);
        if(sum > LIGHTGRAY*4-2)
                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);


}

/*********************************************************************
 *
 *      SETS ALL ENVIRONMENT SPACE TO OCCUPIED AREA.
 *
 *********************************************************************/
void    ZZ_DrawGrayArea(void)
```

```
{
/*      setfillstyle(SOLID_FILL,LIGHTGRAY);
        bar(LEFTLIMIT+10,TOPLIMIT+10,RIGHTLIMIT-10,BOTTOMLIMIT-10);
*/}


/*************************************************************************
 *
 *      DRAW FREE SPACE UNDER LOCATION OF ROBOT
 *
 *************************************************************************/
void    ZZ_DrawBlackBox(void)
{
        int poly[8];
        poly[0]=CarXc[0]+*Xconvert;
        poly[1]=CarYc[0]+*Yconvert;
        poly[2]=CarXc[1]+*Xconvert;
        poly[3]=CarYc[1]+*Yconvert;
        poly[4]=CarXc[2]+*Xconvert;
        poly[5]=CarYc[2]+*Yconvert;
        poly[6]=CarXc[3]+*Xconvert;
        poly[7]=CarYc[3]+*Yconvert;
        setcolor(BLACK);
        setfillstyle(SOLID_FILL,BLACK);
        fillpoly(4,poly);
}
```

```
/*****************************************************************************
 FILE            : ZZ_MISC.H

 DESCRIPTION     : MISCELLANEOUS FUNCTION'S FILE FOR AUTONOMOUS VEHICLE

                   This file contains header file code for assorted functions
                   used throughth out the program.
-----------------------------------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE     : APRIL 27, 1993
*****************************************************************************/
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_MISC.C

 DESCRIPTION  : MISCELLANEOUS FUNCTION'S FILE FOR AUTONOMOUS VEHICLE

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

 by              : Ricardo Andujar

 LAST UPDATE  : APRIL 27, 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_Atan2(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));
}
```

```
/*****************************************************************************
 FILE            : ZZ_NAV.H

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

 by              : Ricardo Andujar

 LAST UPDATE  : APRIL 27, 1993
*****************************************************************************/
void     ZZ_NavLoop(void);
void ZZ_NavServer(byte SourceAddress,int *DataContent,
                                    double *Data, byte *DataNum);
```

```
/*****************************************************************************
 FILE            : ZZ_NAV.C

 DESCRIPTION  : MAIN FILE FOR PROPULSION MODULE OF AUTONOMOUS VEHICLE
 ----------------------------------------------------------------------------
 by              : Ricardo Andujar

 LAST UPDATE  : APRIL 27, 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 QUITING 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;
                }
        }
}
```

```
/*****************************************************************************
FILE            : ZZ_OBJCT.H

DESCRIPTION  : MOVING OBJECT SIMULATION FILE FOR AUTONOMOUS VEHICLE

                  This file contains header file code to display moving objects.
-----------------------------------------------------------------------------

by              : Ricardo Andujar

LAST UPDATE  : APRIL 27, 1993
*****************************************************************************/
void    ZZ_CreateCircle(int x,int y);
void    ZZ_DestroyCircle(void);
void    ZZ_EraseCircle(void);
void    ZZ_DrawCircle(void);
```

```
/*************************************************************************
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  : APRIL 27, 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);
}
```

```
/*****************************************************************************
FILE            : ZZ_PLANR.C

DESCRIPTION  : CONTAINS  FUNCTIONS TO GENERATE MAPPED PATH PLANNER

-----------------------------------------------------------------------------

by              : Ricardo Andujar

LAST UPDATE  : APRIL 27, 1993
*****************************************************************************/
#include<graphics.h>
#include<alloc.h>

#include"ZZ_QUEUE.H"

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

#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;



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;

        /***
         *** Set Target as final destination
         ***/
        PTarget.x = tx;
        PTarget.y = ty;

        /***
         *** Set current position as initial point
         ***/
        Ccw.north = 1;
        Ccw.south = 1;
        Ccw.west = 1;
        Ccw.east = 1;
        Ccw.northeast = 1;
        Ccw.southeast = 1;
        Ccw.northwest = 1;
        Ccw.southwest = 0;

        Cw.north = 1;
        Cw.south = 1;
        Cw.west = 1;
        Cw.east = 1;
        Cw.northeast = 1;
        Cw.southeast = 1;
        Cw.northwest = 1;
        Cw.southwest = 0;
        PCw.x = x;
        PCw.y = y;
        enqueue(&QMain,&PCw);

        while(1)
        {
/*****************************************************************************
*
*                                BASIC PLANNER
*
*****************************************************************************/
        result = lineofsight(&QMain,&PTarget,&Hx,&Hy,&Lx,&Ly);
        if(result == 0 || result == 2)
        {
                PCcw.x = Hx;
                PCcw.y = Hy;
                Cw.x = Hx;
                Cw.y = Hy;
                Ccw.x = Hx;
                Ccw.y = Hy;
                enqueue(&QMain,&PCcw);
                enqueue(&QCw,&PCcw);
                enqueue(&QCcw,&PCcw);
        }
        else
```

```
                break;

/*****************************************************************************
*
*               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);
        }
        /***
         *** 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(getpixel(p->x-j,p->y-j) == WHITE ||
                  getpixel(p->x-j,p->y) == WHITE ||
                  getpixel(p->x,p->y-j) == WHITE ||
                  getpixel(p->x-j,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y) == LIGHTGRAY ||
                  getpixel(p->x,p->y-j) == LIGHTGRAY)
                {
                        p->north = 1;
                        p->northwest = 0;
                        p->x += precision;
                        p->y += precision;
                        return;
```

```
            }
            p->y -= precision;
            for(j=0;j<=box;j+=4)
            if(getpixel(p->x,p->y-j) == WHITE ||
              getpixel(p->x+j,p->y-j) == WHITE ||
              getpixel(p->x-j,p->y-j) == WHITE ||
              getpixel(p->x,p->y-j) == LIGHTGRAY ||
              getpixel(p->x+j,p->y-j) == LIGHTGRAY ||
              getpixel(p->x-j,p->y-j) == LIGHTGRAY)
                    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(getpixel(p->x-j,p->y) == WHITE ||
              getpixel(p->x-j,p->y+j) == WHITE ||
              getpixel(p->x-j,p->y-j) == WHITE ||
              getpixel(p->x-j,p->y) == LIGHTGRAY ||
              getpixel(p->x-j,p->y+j) == LIGHTGRAY ||
              getpixel(p->x-j,p->y-j) == LIGHTGRAY)
            {
                    p->x+=precision;
                    p->northwest = 1;
                    p->west = 0;
                    return;
            }
            p->x -= precision;
            p->y -= precision;
            for(j=0;j<=box;j+=4)
            if(getpixel(p->x-j,p->y-j) == WHITE ||
              getpixel(p->x-j,p->y) == WHITE ||
              getpixel(p->x,p->y-j) == WHITE ||
              getpixel(p->x-j,p->y-j) == LIGHTGRAY ||
              getpixel(p->x-j,p->y) == LIGHTGRAY ||
              getpixel(p->x,p->y-j) == LIGHTGRAY)
            {
                    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(getpixel(p->x-j,p->y+j) == WHITE ||
                  getpixel(p->x-j,p->y) == WHITE ||
                  getpixel(p->x,p->y+j) == WHITE ||
                  getpixel(p->x-j,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y) == LIGHTGRAY ||
                  getpixel(p->x,p->y+j) == LIGHTGRAY)
                {
                        p->west = 1;
                        p->southwest = 0;
                        p->y -= precision;
                        p->x += precision;
                        return;
                }
                p->x -= precision;
                for(j=0;j<=box;j+=4)
                if(getpixel(p->x-j,p->y) == WHITE ||
                  getpixel(p->x-j,p->y+j) == WHITE ||
                  getpixel(p->x-j,p->y-j) == WHITE ||
                  getpixel(p->x-j,p->y) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y-j) == LIGHTGRAY)
                        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(getpixel(p->x,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y+j) == WHITE ||
                  getpixel(p->x-j,p->y+j) == WHITE ||
                  getpixel(p->x,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y+j) == LIGHTGRAY)
                {
                        p->southwest = 1;
                        p->south = 0;
                        p->y -= precision;
                        return;
                }
```

```
                p->x -= precision;
                p->y += precision;
                for(j=0;j<=box;j+=4)
                if(getpixel(p->x-j,p->y+j) == WHITE ||
                  getpixel(p->x-j,p->y) == WHITE ||
                  getpixel(p->x,p->y+j) == WHITE ||
                  getpixel(p->x-j,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y) == LIGHTGRAY ||
                  getpixel(p->x,p->y+j) == LIGHTGRAY)
                {
                        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(getpixel(p->x+j,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y) == WHITE ||
                  getpixel(p->x,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y) == LIGHTGRAY ||
                  getpixel(p->x,p->y+j) == LIGHTGRAY)
                {
                        p->south = 1;
                        p->southeast = 0;
                        p->x -= precision;
                        p->y -= precision;
                        return;
                }
                p->y+=precision;
                for(j=0;j<=box;j+=4)
                if(getpixel(p->x,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y+j) == WHITE ||
                  getpixel(p->x-j,p->y+j) == WHITE ||
                  getpixel(p->x,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y+j) == LIGHTGRAY)
                        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(getpixel(p->x+j,p->y) == WHITE ||
                  getpixel(p->x+j,p->y-j) == WHITE ||
                  getpixel(p->x+j,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y+j) == LIGHTGRAY)
                {
                        p->x -= precision;
                        p->southeast = 1;
                        p->east = 0;
                        return;
                }
                p->x += precision;
                p->y += precision;
                for(j=0;j<=box;j+=4)
                if(getpixel(p->x+j,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y) == WHITE ||
                  getpixel(p->x,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y) == LIGHTGRAY ||
                  getpixel(p->x,p->y+j) == LIGHTGRAY)
                {
                        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(getpixel(p->x+j,p->y-j) == WHITE ||
                  getpixel(p->x,p->y-j) == WHITE ||
                  getpixel(p->x+j,p->y) == WHITE ||
                  getpixel(p->x+j,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y) == LIGHTGRAY)
                {
                        p->x -= precision;
                        p->y += precision;
```

```
                p->east = 1;
                p->northeast = 0;
                return;
        }
        p->x+=precision;
        for(j=0;j<=box;j+=4)
        if(getpixel(p->x+j,p->y) == WHITE ||
          getpixel(p->x+j,p->y-j) == WHITE ||
          getpixel(p->x+j,p->y+j) == WHITE ||
          getpixel(p->x+j,p->y) == LIGHTGRAY ||
          getpixel(p->x+j,p->y-j) == LIGHTGRAY ||
          getpixel(p->x+j,p->y+j) == LIGHTGRAY)
                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(getpixel(p->x,p->y-j) == WHITE ||
                  getpixel(p->x+j,p->y-j) == WHITE ||
                  getpixel(p->x-j,p->y-j) == WHITE ||
                  getpixel(p->x,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y-j) == LIGHTGRAY)
                {
                        p->y += precision;
                        p->northeast = 1;
                        p->north = 0;
                        return;
                }
                p->x += precision;
                p->y -= precision;
                for(j=0;j<=box;j+=4)
                if(getpixel(p->x+j,p->y-j) == WHITE ||
                  getpixel(p->x+j,p->y) == WHITE ||
                  getpixel(p->x,p->y-j) == WHITE ||
                  getpixel(p->x+j,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y) == LIGHTGRAY ||
                  getpixel(p->x,p->y-j) == LIGHTGRAY)
                {
                        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(getpixel(p->x+j,p->y+j) == WHITE ||
                  getpixel(p->x,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y) == WHITE ||
                  getpixel(p->x+j,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x,p->y+j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y) == LIGHTGRAY)
                {
                        p->east = 1;
                        p->southeast = 0;
                        p->x -= precision;
                        p->y -= precision;
                        return;
                }
                p->x+=precision;
                for(j=0;j<=box;j+=4)
                if(getpixel(p->x+j,p->y) == WHITE ||
                  getpixel(p->x+j,p->y-j) == WHITE ||
                  getpixel(p->x+j,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y+j) == LIGHTGRAY)
                        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(getpixel(p->x,p->y+j) == WHITE ||
                  getpixel(p->x+j,p->y+j) == WHITE ||
                  getpixel(p->x-j,p->y+j) == WHITE ||
```

```
            getpixel(p->x,p->y+j) == LIGHTGRAY ||
            getpixel(p->x+j,p->y+j) == LIGHTGRAY ||
            getpixel(p->x-j,p->y+j) == LIGHTGRAY)
        {
                p->southeast = 1;
                p->south = 0;
                p->y -= precision;
                return;
        }
        p->x+=precision;
        p->y+=precision;
        for(j=0;j<=box;j+=4)
        if(getpixel(p->x+j,p->y+j) == WHITE ||
            getpixel(p->x,p->y+j) == WHITE ||
            getpixel(p->x+j,p->y) == WHITE ||
            getpixel(p->x+j,p->y+j) == LIGHTGRAY ||
            getpixel(p->x,p->y+j) == LIGHTGRAY ||
            getpixel(p->x+j,p->y) == LIGHTGRAY)
        {
                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(getpixel(p->x-j,p->y+j) == WHITE ||
                    getpixel(p->x,p->y+j) == WHITE ||
                    getpixel(p->x-j,p->y) == WHITE ||
                    getpixel(p->x-j,p->y+j) == LIGHTGRAY ||
                    getpixel(p->x,p->y+j) == LIGHTGRAY ||
                    getpixel(p->x-j,p->y) == LIGHTGRAY)
                {
                        p->south = 1;
                        p->southwest = 0;
                        p->x += precision;
                        p->y -= precision;
                        return;
                }
                p->y+=precision;
                for(j=0;j<=box;j+=4)
                if(getpixel(p->x,p->y+j) == WHITE ||
                    getpixel(p->x+j,p->y+j) == WHITE ||
                    getpixel(p->x-j,p->y+j) == WHITE ||
                    getpixel(p->x,p->y+j) == LIGHTGRAY ||
```

```
            getpixel(p->x+j,p->y+j) == LIGHTGRAY ||
            getpixel(p->x-j,p->y+j) == LIGHTGRAY)
                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(getpixel(p->x-j,p->y) == WHITE ||
              getpixel(p->x-j,p->y+j) == WHITE ||
              getpixel(p->x-j,p->y-j) == WHITE ||
              getpixel(p->x-j,p->y) == LIGHTGRAY ||
              getpixel(p->x-j,p->y+j) == LIGHTGRAY ||
              getpixel(p->x-j,p->y-j) == LIGHTGRAY)
            {
                    p->southwest = 1;
                    p->west = 0;
                    p->x += precision;
                    return;
            }
            p->x -= precision;
            p->y += precision;
            for(j=0;j<=box;j+=4)
            if(getpixel(p->x-j,p->y+j) == WHITE ||
              getpixel(p->x,p->y+j) == WHITE ||
              getpixel(p->x-j,p->y) == WHITE ||
              getpixel(p->x-j,p->y+j) == LIGHTGRAY ||
              getpixel(p->x,p->y+j) == LIGHTGRAY ||
              getpixel(p->x-j,p->y) == LIGHTGRAY)
            {
                    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(getpixel(p->x-j,p->y-j) == WHITE ||
          getpixel(p->x-j,p->y) == WHITE ||
          getpixel(p->x,p->y-j) == WHITE ||
          getpixel(p->x-j,p->y-j) == LIGHTGRAY ||
          getpixel(p->x-j,p->y) == LIGHTGRAY ||
          getpixel(p->x,p->y-j) == LIGHTGRAY)
        {
                p->west = 1;
                p->northwest = 0;
                p->x += precision;
                p->y += precision;
                return;
        }
        p->x -= precision;
        for(j=0;j<=box;j+=4)
        if(getpixel(p->x-j,p->y) == WHITE ||
          getpixel(p->x-j,p->y+j) == WHITE ||
          getpixel(p->x-j,p->y-j) == WHITE ||
          getpixel(p->x-j,p->y) == LIGHTGRAY ||
          getpixel(p->x-j,p->y+j) == LIGHTGRAY ||
          getpixel(p->x-j,p->y-j) == LIGHTGRAY)
                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(getpixel(p->x,p->y-j) == WHITE ||
                  getpixel(p->x+j,p->y-j) == WHITE ||
                  getpixel(p->x-j,p->y-j) == WHITE ||
                  getpixel(p->x,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x+j,p->y-j) == LIGHTGRAY ||
                  getpixel(p->x-j,p->y-j) == LIGHTGRAY)
                {
                        p->y += precision;
                        p->northwest = 1;
                        p->north = 0;
                        return;
                }
                p->x -= precision;
                p->y -= precision;
                for(j=0;j<=box;j+=4)
```

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

```
/*****************************************************************************
FILE            : ZZ_QUEUE.H

DESCRIPTION     : CONTAINS QUEUE HEADER FILE CODE USED BY
                  THE PATH PLANNER.

-----------------------------------------------------------------------------

by              : Ricardo Andujar

LAST UPDATE     : APRIL 27, 1993
*****************************************************************************/
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_QUEUE.C

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

 -------------------------------------------------------------------------

 by              : Ricardo Andujar

 LAST UPDATE     : APRIL 27, 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


/*************************************************************************
 *
 *      RETURNS A 1 IF QEUEU 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);
}


/*****************************************************************************
 *
 *      ENQUEUS *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(getpixel(x1+j,y1+j) == WHITE ||
                      getpixel(x1+j,y1-j) == WHITE ||
                      getpixel(x1-j,y1+j) == WHITE ||
                      getpixel(x1-j,y1-j) == WHITE ||
                      getpixel(x1+j,y1+j) == LIGHTGRAY ||
                      getpixel(x1+j,y1-j) == LIGHTGRAY ||
                      getpixel(x1-j,y1+j) == LIGHTGRAY ||
                      getpixel(x1-j,y1-j) == LIGHTGRAY)
                      {
                              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 UNNECESARY
 *      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(getpixel(x1+j,y1+j) == WHITE ||
                          getpixel(x1+j,y1-j) == WHITE ||
                          getpixel(x1-j,y1+j) == WHITE ||
                          getpixel(x1-j,y1-j) == WHITE ||
                          getpixel(x1+j,y1+j) == LIGHTGRAY ||
                          getpixel(x1+j,y1-j) == LIGHTGRAY ||
                          getpixel(x1-j,y1+j) == LIGHTGRAY ||
                          getpixel(x1-j,y1-j) == LIGHTGRAY)
```

```
                {
                        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);
}
```

```
/*****************************************************************************
FILE            : ZZ_SUPR2.H

DESCRIPTION  : SUPERVISOR MODULE FOR AUTONOMOUS VEHICLE

                This file contains header file code for the fuzzy controller,
                and high level reasoning.
-----------------------------------------------------------------------------

by              : Ricardo Andujar

LAST UPDATE  : APRIL 27, 1993
*****************************************************************************/
#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_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  : APRIL 27, 1993
*************************************************************************/


/*************************************************************************
 INCLUDE FILES FOR SUPERVISOR MODULE
 *************************************************************************/
#include<alloc.h>
#include<math.h>
#include<stdlib.h>
#include<graphics.h>
#include"ZZ_FUZZY.H"
#include"ZZ_CAN.H"
#include"ZZ_CARSP.H"
#include"ZZ_MISC.H"
#include"ZZ_MAP.H"
#include"ZZ_GRAPH.H"
#include"ZZ_SUPR2.H"
#include"ZZ_QUEUE.H"

#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;

extern void
        ZZ_SupertoMap(double *, double *,double *,
                      double *,double *,double *,double *,int *,int *),
        ZZ_SupertoConsl(double *,double *, int *);



static    double  WayX=1.0,WayY=4.0;
static double TargetX=1.0,TargetY=4.0;
int NewTarget=FALSE;
```

```
/******** INPUT LINGUISTIC VARIABLE (MEMBERSHIP FUNCTION DEF.) ******/
static   ZZ_TRAPEZOID

        BEHINDCAR={160,180,180,200},        /* degrees */
        FRONTOFCAR={-40,0,0,40},
        RIGHTOFCAR={0,80,100,180},
        LEFTOFCAR={-180,-100,-80,-0},

        SMALLDISTANCE={-.5,0,0,.5},  /* meters/sec */
        MEDIUMDISTANCE={0,1,1,2},
        LONGDISTANCE={1,2,7,6},


/********** OUTPUT LINGUISTIC VARIABLES (MEMBERSHIP FUNCTION DEF.) ******/
        STOP={-.45,0,0,.45},            /* meters/sec */
        FWD_SLOW={0,.7,.7,1.4},
        FWD_MEDIUM={0,1,1,2},
        REV_SLOW={-1,-.5,-.5,0},
        REV_MEDIUM={-2,-1,-1,-0},

        STRAIGHT={-30,0,0,30},          /* degrees */
        HARDLEFT={-70,-45,-45,-20},
        HARDRIGHT={20,45,45,70},



        FORWARD={0,0,2,1},
        BACKWARD={-1,-2,0,0},
        LEFT={-22.5,-45,0,0},
        RIGHT={0,0,45,22.5},
        TOO_CLOSE={.1,0,.1,.35},
        VERY_CLOSE={.1,.3,.3,.8},
        CLOSE={0,1,1,1.4},
        GETTING_CLOSER={-1,-2,-.2,0},
        ZERO={-1,0,0,1},
        REVERSE={-2.3,-1.3,-1.3,-.3},
        REVERSE2={-2.6,-1.6,-1.6,-.6},
        GOFORWARD={-2.6,-1.6,-1.6,-.6},
        SLOWDOWN={-1.9,-.9,-.9,.1},
        SLOWABIT={-1.7,-.7,-.7,.3},
        TO_LEFT={-20,-14,-16,-10},
        TO_RIGHT={10,16,14,20},
        HARDLEFT2={-65,-65,-25,-25},
        HARDRIGHT2={25,25,65,65}


        ;

/********** DISCRETE FUZZY OUTPUTS **********/
static   ZZ_FUZZYOUTPUT

        FRoadSpeed,
        FSteer,
        FSpeedChange,
        FSteeringChange;
```

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

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);
        ZZ_SupertoConsl(&WayX,&WayY,&NewTarget);

        ZZ_InitGraph();
        ZZ_InitMap();

        ZZ_InitFuzzyOutput(-65,65,&FSteer,20);
        ZZ_InitFuzzyOutput(-6,6,&FRoadSpeed,20);
        ZZ_InitFuzzyOutput(-80,80,&FSteeringChange,30);
        ZZ_InitFuzzyOutput(-3,3,&FSpeedChange,20);
        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;
}


/*****************************************************************************
 *
 *      This Function should be called when terminating
 *      program execution
 *
 *****************************************************************************/
void    ZZ_EraseSuper(void)
{
        ZZ_CloseGraph();
        ZZ_DelFuzzyOutput(&FSteer);
        ZZ_DelFuzzyOutput(&FRoadSpeed);
}


/*****************************************************************************
 *
 *       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 =60.0l*modf(timee/60.0l,&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
 *   acually implemented.  *
 *
 *****************************************************************************/
void    ZZ_GetSensorData(void)
{
        int i=1;
        ZZ_CAN_Request(HIGHPRIORITY,SUPERVISOR,VISION,&DataContent,
                                                        Range,&DataNum);


        ZZ_CAN_Request(HIGHPRIORITY,SUPERVISOR,NAVIGATION,&DataContent,
                                                        Data,&DataNum);
        i=1;
        ARoadSpeed      = Data[i++];
        Compass         = Data[i++];
        Xposition       = Data[i++];
        Yposition       = Data[i++];
        ASteeringAngle  = Data[i++];
```

```
        ABrake          = Data[i++];

        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;
                }
                /***
                 *** Otherwise, generate a new path
                 *** and use the first point as the current
                 *** waypoint.
                 ***/
                else
                        NewTarget = TRUE;
        }

        /***
         ***     New Path has been generated or new target has been selected
         ***/
        if(NewTarget)
        {
        /*** Generate new path and set current waypoint as first point
         *** in new path
         ***/
                if(NewTarget == FIRST_TIME)
                {
                        gotoxy(1,1);
                        printf("TIME : %4.0lf minutes, %2d seconds",0.0l,0);
                        TargetX = WayX;
```

```
                TargetY = WayY;
                timee = 0;
                timer1 = 0;
                METHOD = PLANNER;
        }
        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)
{
        double  min,
                AngleAdj;

/******* CONVERT Waypointangle from radians to degrees      ******/
        WayPointAngle *= 180.0/M_PI;


/******* RESET fuzzy output variables                ******/
        ZZ_ClearFuzzyOutput(&FSteer);
        ZZ_ClearFuzzyOutput(&FRoadSpeed);


/******* AngleAdj is used only by BEHIND CAR Rules          ******/
        if(WayPointAngle<0)
                AngleAdj=WayPointAngle+360.0;
        else
                AngleAdj = WayPointAngle;


/*****************************************************************************
 ******          FUZZY INFERENCE RULES  USING MIN/MAX RULES
 *****************************************************************************/

/**********          Waypoint is in front of robot          **********/
        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&FRONTOFCAR),
           ZZ_Member(WayPointDist,&SMALLDISTANCE)))
        {
                ZZ_AddMax(min,&STOP,&FRoadSpeed);
                ZZ_AddMax(min,&STRAIGHT,&FSteer);
        }
        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&FRONTOFCAR),
           ZZ_Member(WayPointDist,&MEDIUMDISTANCE)))
        {
                ZZ_AddMax(min,&FWD_SLOW,&FRoadSpeed);
                ZZ_AddMax(min,&STRAIGHT,&FSteer);
        }

        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&FRONTOFCAR),
           ZZ_Member(WayPointDist,&LONGDISTANCE)))
        {
                ZZ_AddMax(min,&FWD_MEDIUM,&FRoadSpeed);
                ZZ_AddMax(min,&STRAIGHT,&FSteer);
```

```
        }

/**********          Waypoint is behind robot     **********/
        ZZ_Swap(&WayPointAngle,&AngleAdj);
        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&BEHINDCAR),
           ZZ_Member(WayPointDist,&SMALLDISTANCE)))
        {
                ZZ_AddMax(min,&STOP,&FRoadSpeed);
                ZZ_AddMax(min,&STRAIGHT,&FSteer);
        }

        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&BEHINDCAR),
                   ZZ_Member(WayPointDist,&MEDIUMDISTANCE)))
        {
                ZZ_AddMax(min,&REV_SLOW,&FRoadSpeed);
                ZZ_AddMax(min,&STRAIGHT,&FSteer);
        }
        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&BEHINDCAR),
           ZZ_Member(WayPointDist,&LONGDISTANCE)))
        {
                ZZ_AddMax(min,&REV_MEDIUM,&FRoadSpeed);
                ZZ_AddMax(min,&STRAIGHT,&FSteer);
        }
        ZZ_Swap(&WayPointAngle,&AngleAdj);


/**********        Waypoint is to the right of the robot          **********/

        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&RIGHTOFCAR),
           ZZ_Member(WayPointDist,&SMALLDISTANCE)))
        {
                ZZ_AddMax(min,&STOP,&FRoadSpeed);
                ZZ_AddMax(min,&HARDRIGHT,&FSteer);
        }
        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&RIGHTOFCAR),
                   ZZ_Member(WayPointDist,&MEDIUMDISTANCE)))
        {
                ZZ_AddMax(min,&FWD_SLOW,&FRoadSpeed);
                ZZ_AddMax(min,&HARDRIGHT,&FSteer);
        }
        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&RIGHTOFCAR),
           ZZ_Member(WayPointDist,&LONGDISTANCE)))
        {
                ZZ_AddMax(min,&FWD_MEDIUM,&FRoadSpeed);
                ZZ_AddMax(min,&HARDRIGHT,&FSteer);
        }


/**********        Waypoint is to the left of the robot     **********/

        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&LEFTOFCAR),
           ZZ_Member(WayPointDist,&SMALLDISTANCE)))
        {
                ZZ_AddMax(min,&STOP,&FRoadSpeed);
```

```
            ZZ_AddMax(min,&HARDLEFT,&FSteer);
        }
        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&LEFTOFCAR),
           ZZ_Member(WayPointDist,&MEDIUMDISTANCE)))
        {
            ZZ_AddMax(min,&FWD_SLOW,&FRoadSpeed);
            ZZ_AddMax(min,&HARDLEFT,&FSteer);
        }

        if(min = ZZ_Min(ZZ_Member(WayPointAngle,&LEFTOFCAR),
              ZZ_Member(WayPointDist,&LONGDISTANCE)))
        {
            ZZ_AddMax(min,&FWD_MEDIUM,&FRoadSpeed);
            ZZ_AddMax(min,&HARDLEFT,&FSteer);
        }

/*****************************************************************************
 **********
 **********      DEFUZZIFICATION OF OUTPUT VARIABLES USING LARSEN'S RULE
 **********
 *****************************************************************************/
        SteeringAngle = ZZ_Defuzzify(&FSteer)*M_PI/180.0;
        RoadSpeed = ZZ_Defuzzify(&FRoadSpeed);
}


/*****************************************************************************
 *
 *  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)
{
        double max,min,SteeringChange=0,SpeedChange=0;

        /***
         *** Clear fuzzy output variables
         ***/
        ZZ_ClearFuzzyOutput(&FSteeringChange);
        ZZ_ClearFuzzyOutput(&FSpeedChange);

        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
         ***/
        if(min=ZZ_Member(min,&CLOSE))
        {
                ZZ_AddMax(min,&SLOWABIT,&FSpeedChange);
        }


        /***
         *** 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);

    }

/***
 ***    FUZZY RULES USED TO AVOID COLLISIONS
 ***/

        /***
         ***        Too close to obstacle
         ***/
        if(GET_OUT == OFF)
```

```
{
if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
          ZZ_Min(ZZ_Member(SteeringAngle,&STRAIGHT),
                ZZ_Member(Front,&TOO_CLOSE))))
{
        ZZ_AddMax(min,&REVERSE2,&FSpeedChange);
}

if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
          ZZ_Min(ZZ_Member(Front,&TOO_CLOSE),
                ZZ_Member(SteeringAngle,&LEFT))))
{
        ZZ_AddMax(min,&REVERSE2,&FSpeedChange);
        ZZ_AddMax(min,&HARDRIGHT,&FSteeringChange);
}

if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
          ZZ_Min(ZZ_Member(Front,&TOO_CLOSE),
                ZZ_Member(SteeringAngle,&RIGHT))))
{
        ZZ_AddMax(min,&REVERSE2,&FSpeedChange);
        ZZ_AddMax(min,&HARDLEFT,&FSteeringChange);
}
if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
          ZZ_Min(ZZ_Member(SteeringAngle,&LEFT),
                ZZ_Member(Left2,&TOO_CLOSE))))
{
        ZZ_AddMax(min,&SLOWDOWN,&FSpeedChange);
        ZZ_AddMax(min,&TO_RIGHT,&FSteeringChange);
}

if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
          ZZ_Min(ZZ_Member(SteeringAngle,&RIGHT),
                ZZ_Member(Right2,&TOO_CLOSE))))
{
        ZZ_AddMax(min,&SLOWDOWN,&FSpeedChange);
        ZZ_AddMax(min,&TO_LEFT,&FSteeringChange);
}
/***
***      very close to obstacle
***/
if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
          ZZ_Min(ZZ_Member(SteeringAngle,&STRAIGHT),
                ZZ_Member(Right,&VERY_CLOSE))))
{
        ZZ_AddMax(min,&SLOWDOWN,&FSpeedChange);
        ZZ_AddMax(min,&HARDRIGHT,&FSteeringChange);
}
if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
          ZZ_Min(ZZ_Member(SteeringAngle,&LEFT),
                ZZ_Member(Right,&VERY_CLOSE))))
{
        ZZ_AddMax(min,&SLOWDOWN,&FSpeedChange);
        ZZ_AddMax(min,&HARDRIGHT,&FSteeringChange);
}
if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
```

```
            ZZ_Min(ZZ_Member(SteeringAngle,&RIGHT),
                    ZZ_Member(Left,&VERY_CLOSE))))
    {
            ZZ_AddMax(min,&SLOWDOWN,&FSpeedChange);
            ZZ_AddMax(min,&HARDLEFT,&FSteeringChange);
    }

    if(min = ZZ_Min(ZZ_Member(RoadSpeed,&FORWARD),
            ZZ_Min(ZZ_Member(SteeringAngle,&STRAIGHT),
                    ZZ_Member(Left,&VERY_CLOSE))))
    {
            ZZ_AddMax(min,&SLOWDOWN,&FSpeedChange);
            ZZ_AddMax(min,&HARDLEFT,&FSteeringChange);
    }

    }
    /***
     ***      REVERSE COLLISION AVOIDANCE
     ***/
    if(min = ZZ_Min(ZZ_Member(RoadSpeed,&BACKWARD),
            ZZ_Min(ZZ_Member(SteeringAngle,&STRAIGHT),
                    ZZ_Member(Back,&TOO_CLOSE))))
    {
            ZZ_AddMax(min,&GOFORWARD,&FSpeedChange);
    }

    if(min = ZZ_Min(ZZ_Member(RoadSpeed,&BACKWARD),
            ZZ_Min(ZZ_Member(Left2,&TOO_CLOSE),
                    ZZ_Member(SteeringAngle,&LEFT))))
    {
            ZZ_AddMax(min,&SLOWDOWN,&FSpeedChange);
            ZZ_AddMax(min,&HARDRIGHT2,&FSteeringChange);
    }

    if(min = ZZ_Min(ZZ_Member(RoadSpeed,&BACKWARD),
            ZZ_Min(ZZ_Member(Right2,&TOO_CLOSE),
                    ZZ_Member(SteeringAngle,&RIGHT))))
    {
            ZZ_AddMax(min,&SLOWDOWN,&FSpeedChange);
            ZZ_AddMax(min,&HARDLEFT2,&FSteeringChange);
    }
    /*****************************************************************
     ***      DEFUZZIFY OUTPUTS                                    ***
     *****************************************************************/
    SteeringAngle*= M_PI/180.0;
    SteeringChange = ZZ_Defuzzify(&FSteeringChange)*M_PI/180.0;
    SpeedChange = ZZ_Defuzzify(&FSpeedChange);
    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);
}
```

```
/*****************************************************************************
FILE            : ZZ_VSION.H

DESCRIPTION  : VISION MODULE FOR AUTONOMOUS VEHICLE

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

by              : Ricardo Andujar

LAST UPDATE  : APRIL 27, 1993
*****************************************************************************/
#define ON 1
#define ROADRANGEC1 1.4
#define ROADRANGEC2 3.2
#define SPEED_SOUND 343.0   /* (meters/sec)  */
#define SCREENCONVERT 40.0  /* (pixel/meter) */
#define SMAXRANGE    400
#define  HALFSPREADANGLE  10.0/180.0*M_PI /*10 Degrees*/
#define NUM_SENSORS 6


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

void     ZZ_VisionLoop(void);
```

```
/*****************************************************************************
 FILE            : ZZ_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  : APRIL 27, 1993
*****************************************************************************/


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


/********************** 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)
{
}
```

VITA 2

Ricardo Andujar

Candidate for the Degree of

Master of Science

Thesis: AUTONOMOUS VEHICLE CONTROL USING FUZZY INFERENCE AND A FAST PATH PLANNING ALGORITHM

Major Field: Mechanical Engineering

Biographical:

Personal Data: Born in Caguas, Puerto Rico, April 1, 1968, the son of Ricardo Andujar Sr. and Felipa Guiterrez.

Education: Graduated from Eloisa Pascual High School, Caguas, Puerto Rico, in May 1986; received Bachelor of Science Degree in Mechanical Engineering from Oklahoma State University at Stillwater in July, 1991; completed requirements for the Master of Science degree at Oklahoma State University in July, 1993.

Professional Experience: Teaching Assistant, Department of Mechanical and Aerospace Engineering, Oklahoma State University, August, 1992, to May, 1993.