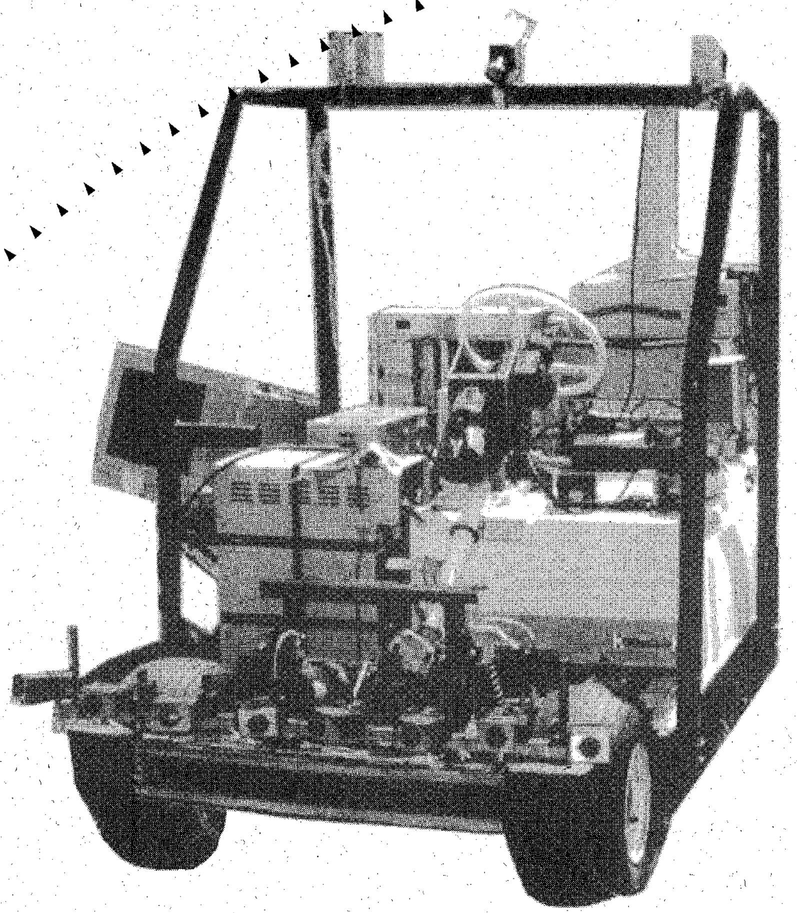
Report Number 97-04

Research



ALX: Autonomous Vehicle Guidance For Roadway Following and Obstacle Avoidance

UNIVERSITY OF MINNESOTA
CENTER FOR TRANSPORTATION STUDIES

3728 0201 C. 3

Technical Report Documentation Page

1. Report No. MN/RC - 97/04	2.	3. Recipient's Accession No.
4. Title and Subtitle ALX: AUTONOMOUS VEHICLE GUIDANCE FOR ROADWAY FOLLOWING AND OBSTACLE AVOIDANCE		5. Report Date October 1996
		6.
7. Author(s) Yu-feng Du, William Schiller, Don Krantz, Craig Shankwitz, and Max Donath		8. Performing Organization Report No.
9. Performing Organization Name and Address University of Minnesota Department of Mechanical Engineering 211 MechE Building 111 Church St S.E. Minneapolis, Minnesota 55455		10. Project/Task/Work Unit No.
		11. Contract (C) or Grant (G) No. (C) 73168 TOC #174
12. Sponsoring Organization Name and Address Minnesota Department of Transportation 395 John Ireland Boulevard Mail Stop 330 St. Paul, Minnesota 55155		13. Type of Report and Period Covered Final Report 1994-1996
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract (Limit: 200 words)

This report presents results of the research performed on the Autonomous Land Experimental Vehicle (ALX) at the University of Minnesota. ALX autonomously follows roadways through the use of visual perception, and executes obstacle detection and collision avoidance through the use of ultrasonic sonar range sensors. This report describes the ALX embedded real-time control system based on a multi-processor, multi-tasking architecture, and presents algorithms used for visual perception, path tracking, position estimation, obstacle detection, and collision avoidance. Computer simulation and experimental results also are presented.

17. Document Analysis/Descriptors Advanced Vehicle Control Systems Obstacle Detection Vision-Based Guidance	Road Following Collision Avoidance	18. Availability Statement No restrictions. Document available from: National Technical Information Services, Springfield, Virginia 22161	
19. Security Class (this report) Unclassified	20. Security Class (this page) Unclassified	21. No. of Pages 139	22. Price

ALX: AUTONOMOUS VEHICLE GUIDANCE FOR ROADWAY FOLLOWING AND OBSTACLE AVOIDANCE

Final Report

Prepared by:

Yu-feng Du
William Schiller
Donald Krantz
Craig Shankwitz
Max Donath

Department of Mechanical Engineering,
the Center for Advanced Manufacturing, Design and Control (CAMDAC)

The University of Minnesota
111 Church Street SE
Minneapolis MN 55455

October 1996

Published by

Minnesota Department of Transportation
Office of Research Administration
200 Ford Building Mail Stop 330
117 University Avenue
St Paul Minnesota 55155

ACKNOWLEDGEMENTS

We would like to thank Jon Minners, Micah Garlich-Miller, Kirt Kvistberg, Vassilios Morellas, Lee Alexander and Curt Olson for their contributions to the ALX project.

TABLE OF CONTENTS

Chapter 1 INTRODUCTION 1

Project Background 1

Related Work 3

Chapter 2 SYSTEM DESIGN 13

System Overview 13

Real-Time Control Architecture 20

VxWorks Real-Time Operating System 20

Cross-Development Environment 22

Multi-Tasking Control Structure 23

Chapter 3 PATH TRACKING 27

Visual Sensing of the Roadway 27

Comparison of Blob vs Edge Detection 27

Blob Analysis 29

Datacube Image Processing System 29

Data Pipelines 30

Connectivity Analysis 35

Blob Association and Curve Fitting 38

Path Generation 39

Path Based on Goals 40

Goal Generation 42

Heading Control Scheme 45

Overall Control Strategy 49

Chapter 4 OBSTACLE DETECTION AND COLLISION AVOIDANCE 53

Spatial Information Sensing 53

Systematic Errors 55

Sensor Information Mapping 58

Multi-Value Labeling Scheme 64

Alternative Path Searching 67

Chapter 5 SIMULATION RESULTS 75

Chapter 6 EXPERIMENTAL RESULTS 83

Roadway Following Test 83

Obstacle Avoidance Test 87

Overall Performance Test 93

Chapter 7 CONCLUSIONS 97

System Evaluation 97

Recommendations 99

Summary 100

REFERENCES 103

APPENDIX A Dead-Reckoning System

Introduction

Magnetometer Calibration

Dead-Reckoning Position Calculation

APPENDIX B Vision Data Transformation

APPENDIX C Vision Algorithm for Road Following

Tuning and Running Blob Analysis

APPENDIX D Program Descriptions

List of Figures

1. Figure 2-1 (a). ALX - Autonomous Land Experimental Vehicle 13
2. Figure 2-1 (b). ALX dimensions 14
3. Figure 2-2. System device overview 15
4. Figure 2-3. Ultrasonic range sensor layout 17
5. Figure 2-4. Ultrasonic range sensing system 18
6. Figure 2-5. Cross development setup 23
7. Figure 2-6. Multi-tasking structure 24
8. Figure 3-1. Simplified flow chart of blob analysis vision algorithm 30
9. Figure 3-2. Threshold and histogram Data Pipeline 32
10. Figure 3-3. Morphological filter, run length encode and chain code data pipeline . . . 32
11. Figure 3-4. Example of a Binary Image(A) and the Corresponding Run Length Encoded Image (B) 33
12. Figure 3-5. Flow chart for vision algorithm - modified 35
13. Figure 3-6. Connectivity analysis flow chart 37
14. Figure 3-7. Binary image of the roadway with previous image lines 39
15. Figure 3-8. Reconstructed roadway edges 39
16. Figure 3-9 (a). Illustration of the goal stack concept and operation 41
17. Figure 3-9 (b). Illustration of the goal stack concept and operation 42
18. Figure 3-10 (a). Goals are generated from two evenly visible roadway edges 43
19. Figure 3-10 (b). Goals are generated from only one visible roadway edge 44
20. Figure 3-10 (c). Goals are generated from two unevenly visible roadway edges 45
21. Figure 3-11. Heading control of ALX 46
22. Figure 3-12. State transition diagram of overall strategy 49
23. Figure 4-1. The Virtual Map 54
24. Figure 4-2. Four possible interpretations of an isolated return from a sensor scan . . 56
25. Figure 4-3. Conflict label assignments 57
26. Figure 4-4. Assignments of sector numbers 59
27. Figure 4-5. Overall bounding rectangle of a sensor beam in sector 1 60
28. Figure 4-6. Calculation of horizontal extent 61
29. Figure 4-7. Consistent labeling. Conflict labels shown in Figure 4.3 67
30. Figure 4-8. Detouring goals are generated during collision avoidance path planning 69

31. Figure 4-9. Binary-tree search method 70
32. Figure 4-10. Apply the binary-tree search method on the virtual map 71
33. Figure 5-1. Output window of Navsim - a simulation program for ALX 76
34. Figure 5-2. Command window of Navsim 77
35. Figure 5-3. A small window displays the overview of the simulated roadway 78
36. Figure 5-4. In simulation, ALX detected an obstacle and calculated an alternative path to go around the obstacle 79
37. Figure 5-5. In simulation, ALX successfully avoided an obstacle and resumed path tracking 80
38. Figure 5-6. In simulation, ALX passed a roadway section which contained a number of obstacles 81
39. Figure 6-1 (a). Typical input image 84
40. Figure 6-1 (b). Typical output image 84
41. Figure 6-2. ALX is able to follow a test roadway 85
42. Figure 6-3. ALX follows a turning section of testing track 86
43. Figure 6-4. ALX obstacle avoidance operation - single obstacle 88
44. Figure 6-5. ALX obstacle avoidance operation - single obstacle 89
45. Figure 6-6. Motion of ALX in response to two obstacles, situation (a) 90
46. Figure 6-7. Motion of ALX in response to two obstacles, situation (b) 91
47. Figure 6-8. ALX performs roadway following with obstacle avoidance operation . . 93
48. Figure 6-9. ALX performs roadway following with obstacle avoidance operation . . 94
49. Figure A-1. Theoretical output of one axis of the fluxgate magnetometer A2
50. Figure A-2. Flowchart of the magnetometer calibration process A4
51. Figure A-3. Generate data for least-square curve fitting A5
52. Figure A-4. Dead-reckoning position calculation A7
53. Figure B-1. Camera model for image data transformation B1
54. Figure B-2. Calculation of the focal length of the camera B3
55. Figure B-3. Calculation of Yw B4
56. Figure D-1. Programs that support multi-tasking control structure D9

List of Tables

1. Table D-1. Program list D2
2. Table D-2. Program List - Vision Algorithm D7

EXECUTIVE SUMMARY

This report presents results of research performed on the Autonomous Land Experimental Vehicle (ALX) at the University of Minnesota. ALX autonomously follows roadways through the use of visual perception, and executes obstacle detection and collision avoidance using a suite of ultrasonic sonar range sensors. This report describes the ALX embedded real-time control system based on a multi-processor, multi-tasking architecture, and presents algorithms used for visual perception, path tracking, position estimation, obstacle detection and collision avoidance. Computer simulation and experimental results are also presented.

A summary of this report entitled "ALX: Autonomous Vehicle Guidance for Roadway Following and Obstacle Avoidance" was presented at the 1995 IEEE Systems, Man and Cybernetics Conference, Vancouver, British Columbia, and is contained in the proceedings (pp. 364-370) published by IEEE.

CHAPTER 1
INTRODUCTION

1.1 Project Background

The Intelligent Transportation Systems (ITS) program underway in the United States promises to radically change the methods with which goods and people are transported throughout the country. These changes will arise from the application of emerging computer control and communication technologies to both vehicles and the transportation infrastructure. The Advanced Vehicle Control Systems (AVCS) represent one very important activity under the ITS umbrella. AVCS are designed to assist the driver with vehicle control tasks such as intelligent speed regulation, obstacle detection and collision avoidance, etc.

The main impetus for the development of AVCS is safety. The National Traffic Safety Board has indicated that in excess of 40% of heavy truck accidents are due to driver fatigue. Technologies that provide for collision avoidance and lane following can potentially "reduce urban and rural freeway accidents by a minimum of 30 and 35 percent, respectively. Accident reductions of this magnitude would eliminate approximately 71,000 accidents per year and $700 million in accidents costs" [1].

With the Minnesota Department of Transportation, we are working on systems capable of taking over control of a semi tractor-trailer when drivers become incapacitated. Such systems must be able to provide for integrated speed and headway regulation, roadway following, obstacle (and other vehicle) detection and collision avoidance. At the minimum, the control system should be able to safely drive the vehicle over to the shoulder (if and where it exists), slow down and come to a complete stop.

In the first phase of the SAFE TRUCK Program, we are focusing on subsystems which prevent road departure accidents. The goal for these subsystems is to keep the vehicle in its lane. The tractor-trailer experimental testbed will use a new differential global positioning system (DGPS) system capable of providing real-time 5 Hz uninterpolated position of the tractor accurate to better than 20 cm circular error probability (CEP). This global positioning system (GPS) is integrated with an inertial measurement unit using a Kalman filter [2]. In subsequent phases, additional lateral guidance systems will be investigated including magnetic striping sensed by vehicle mounted magnetometers.

For safety reasons, it is important to test potentially high risk, innovative vehicle sensing and control strategies on small vehicles before implementation on the semi tractor-trailer. We designed the Autonomous Land Experimental Vehicle (ALX) to provide such a safe platform.

ALX is based on an electric golf cart chassis and is equipped with control computers, actuators and feedback sensors, dead-reckoning sensors for position estimation, an array of up to sixteen ultrasonic range sensors with overlapping fields for obstacle detection, and a vision system for roadway sensing. It was designed to solve a specific problem: navigating an *a priori* unknown outdoor road with paint-stripe lane boundaries, under varying lighting conditions, while avoiding obstacles of unknown size, shape, number, and placement.

This paper provides the current research status of the ALX project. It documents various systems built for ALX, describes the embedded real-time control system based on a multi-tasking architecture, and presents algorithms used for visual perception, path tracking, position estimation, obstacle detection and collision avoidance. Computer simulation and experimental results are also presented. The report concludes with a description of the next phase of development and recommendations to further extend ALX control schemes for use on a full-scale commercial vehicle for highway operation.

The remainder of this section reviews some of the previous work done in areas related to autonomous land vehicles and the approach we used in designing ALX.

1.2 Related Work

Navigation and collision avoidance constitute the primary objectives of the driving task, and also account for the majority of the effort required to develop an autonomous vehicle control system. Before starting the system design of ALX, many of the existing systems described in the literature were evaluated. Some of the systems studied included:

- Lumelsky's system [3]. Like many robot systems for dealing with unknown environments, the problem is to completely map a finite scene (the mapping problem). However, the immediate objective of ALX is not complete mapping, but determination and execution of paths through an a priori unknown environment.

- Kamada's system [4]. This represents a class of solutions that assumes known and cooperative environments. Kamada, for example, uses signpost markers to update position relative to a known map. (It is interesting to note that the great majority of the research done on autonomous robots deals with known environments.)

- Researchers at the Georgia Institute of Technology have applied vision-based landmark tracking to an Automated Guided Vehicle (AGV). However, the system used artificial landmarks (circular pieces of retroreflective paper) illuminated by a strobe on the vehicle [5]. This facilitated the camera threshold adjustment so that only the landmark was visible.

- At the University of Maryland, Dickingson and Davis [6] developed an autonomous robotic vehicle which uses landmarks to estimate vehicle position. This system is not appropriate for ALX because no landmarks will be available in the a priori unknown outdoor environment.

- At Carnegie Mellon University, Thorpe [7,8] has worked on the road-following problem, which is an essential part of ALX requirement.

  Thorpe's newer work [8], described also in [9], uses neural networks and a vision system to pilot a vehicle (called NavLab). A laser range finder is used for obstacle detection. The limitation of Thorpe's system is that it needs to be re-trained for each road type; it cannot presently be trained once to cover many different road types. This method was also rejected because we do not have the computer resources to implement such a system. The computing resources for the NavLab consist of five Sun microcomputers interconnected via EtherNet, working together with Warp, a specially designed multiprocessor.

- Dickmanns and other German researchers have developed an autonomous Mercedes van which can operate on a well-structured freeway [10]. In this system, vision information alone is used for collision avoidance, navigation, and guidance. One problem with this approach is that the number of obstacles tracked is limited by the vision processors on board of the vehicle; in the reference cited, no more than three obstacles can be tracked at any one time.

The survey of the literature shows that although there are many working systems which are capable of conducting obstacle detection and collision avoidance for autonomous robotic vehicles, many of them are limited because of reliance on known environment, or lack of lateral control (roadway following) functionality.

The following issues were advised during ALX's system design process.

a) Real-Time Computing Architecture

In classical sequential control, system actions are strictly ordered as a predefined sequence in time: the behavior of the program depends only on the effects of the individual actions and their sequential order; the time at which action is taken is not of consequence. Traditional sequential control systems are not appropriate for applications such as the autonomous navigation of a vehicle because of the ongoing occurrence of multiple continuous and discrete events in the environment around the vehicle. Responding with actions in a pre-defined order usually will not satisfy time constraints for successful vehicle control.

Unlike the Navlab vehicle which uses a distributed computing architecture [8], we chose to use the VxWorks Real-Time Operating System (RTOS) to construct our computing architecture. By using VxWorks, we were able to develop an embedded real-time control system which differs from traditional control systems in that the sequence of actions is not determined by the designer but by events occurring in real-time in the outside world.

b) Vision System

The vision system (including cameras, lenses, processing computers, algorithms, etc.) is considered the most complicated subsystem of an autonomous vehicle for outdoor operation. This is largely because of the wide variety of roadway scenarios and illumination conditions.

There are many algorithms that have been developed for roadway following. For example, at Carnegie Mellon University, several different algorithms were developed for following an unstructured road. These methods include supervised and unsupervised classification approaches for color vision [11], an explicit-model-based approach [12], and a knowledge-based interpretation of road scenes approach [13]. In Germany, Dickmanns used "Regions of Interest" (ROI) to process each image frame obtained to find the edges of a structured freeway [10]. In both CMU and Dickmanns' work, specially designed image processors were used.

Because ALX is designed to operate only on structured roadways which are delimited by painted lane markers, the vision system on ALX is less sophisticated than that used by the NavLab at CMU. ALX vision system is responsible for visual perception of the roadway by detection of the road edges. A generic vision algorithm which uses techniques including optimized thresholding, blob detection and association, and roadway edge construction and projection has been developed and tested. Through a series of tests, the performance of the vision system was evaluated. The vision system guided ALX through a test track consisting of both straight sections and corners of varying curvatures under bright sunlight, under overcast conditions, and in low light intensity situations (i.e., dusk and dawn).

c) Dead-Reckoning System

Dead-reckoning sensors are used to construct a position estimator for autonomous robotic vehicles. The term "dead reckoning navigation" refers to a mathematical process of projecting a point from a known starting location into a Cartesian coordinate space. The heading of the robot is most easily measured directly, and the most cost-effective solution for outdoor ground vehicles is the fluxgate magnetometer. These sensors use a wire-wound toroid to sense the earth's magnetic field and typically give data accurate to about one degree. They are relatively inexpensive (less than $1,000) and are available commercially from many sources. More detailed information can be found in a report from the Sandia National Laboratory [15].

The dead-reckoning system for ALX uses on-board dead-reckoning sensors including a fluxgate magnetometer and a wheel encoder. It is responsible for estimating the vehicle's world position [in (x, y) coordinates relative to the starting position] and its orientation (azimuth with respect to relative north). Other entities sensed by ALX (road edges, obstacles) are referenced to this coordinate system. Since this is a dead-reckoning system, the certainty of the coordinate's accuracy decays with distance moved after measurement. A differential-GPS (global position system) is being developed and will be used on ALX to conduct periodic position estimation corrections [2].

d) Range Sensor

The range sensor is responsible for detecting obstacles in the local environment. Due to recent advances in MIMIC (Millimeter Monolithic Integrated Circuit) radar technology, it has been proposed that MIMIC radar be used for the range finder on the semi tractor-trailer controller which is being developed at the University of Minnesota [2]. However, at the time when the ALX project began, off-the-shelf MIMIC radar product with a reasonable performance/cost ratio were not available. Under this circumstance, ultrasonic range sensing was chosen for ALX.

Ultrasonic range sensors utilize high-frequency pressure waves to detect the presence of and distance to objects. They have been used extensively on indoor autonomous vehicles. However, it has been reported that in outdoor environments, ultrasonic range measurements are adversely affected by weather factors such as a strong head-wind. A driving rain may cause reflections that could render the system useless [15]. Still, the low cost of each sensor suite made the ultrasonic range sensor attractive for initial obstacle detection and collision avoidance algorithm test.

One problem encountered while using the ultrasonic range-range sensors was a variety of systematic errors similar to those documented by Beckerman [19] and Kuc [20]. Systematic errors are those errors that mainly result from incorrect and inconsistent interpretations of the raw ultrasonic range data during processing. This is largely due to the relatively wide beam width and low resolution of ultrasonic transducers, which cause problems when attempting to characterize the local environment of the robot. For instance, the standard Polaroid ultrasonic range-ranger has an effective beam width between 12 to 15 degrees. The sensor footprint is the intersection of a cone and a sphere, with the cone's origin coincident with the sphere's center. The sphere's radius is the range-to-obstacle. The obstacle itself can be anywhere on the surface of the sphere within the cone. The volume enclosed by the intersection of the cone and sphere is likely to be unoccupied.[1] The environmental geometry and surface properties of the objects being sensed can also contribute to

[1] Specular reflection can make this assumption untrue For details refer to [19] or [20].

systematic errors. By using multiple sensors with overlapping beams, the angular resolution can be increased. On ALX, ultrasonic range-sensors are purposely mounted such that their footprints overlap. The obstacle detection algorithm also exploits the movement of the platform to refine the size and position estimates of obstacles.

e) Mapping Scheme

An autonomous mobile robot should have the ability to construct maps of its environment and to find a path to its destination. We have developed a mapping scheme based on the concept of *virtual map*. The virtual map is a Cartesian map which is constructed with a 2-D array of *cells*. It stores sensed information about the environment around ALX into this cell array which moves along with the robot. The virtual map is a discrete representation of the robot's local environment. Its purpose is to determine the local trafficability, i.e., if a desired path is obstructed, the virtual map is used to search for a new path. For successful operation, the virtual map should contain accurate and clearly delineated open spaces and obstacles. However, because of various kinds of uncertainty in sensor information in the real world, perfect knowledge about the environment is very difficult to obtain.

To remove systematic ultrasonic range sensor error and store information from ultrasonic range-range sensors in the form of Cartesian maps, many stochastic methodologies have been developed by researchers at Carnegie-Mellon University ([21], [22], and [23]) and at Oak Ridge National Laboratory [24] . The CMU methods used probability distributions, and Fryxell [24] used a voting procedure to represent initial information on occupied and empty regions of space. In both approaches, cumulative information, stored in separate occupied and empty maps, was used to generate a binary navigation map. To remove the systematic errors, instead of using stochastic schemes such as the two mentioned above, we used a multivalued labeling scheme to tag the data as it was registered into one map. We were motivated by the observation that it would be useful to have simple, nonstochastic methods for the identification and removal of systematic errors arising from the processing of sparse sensor data. This method is an extension of the labeling concept developed by Beckerman and Oblow [19] at Oak Ridge National Laboratories.

The main goal of the labeling algorithm is to remove ambiguities of obstacle position caused by the ultrasonic range sensor's finite footprint and to remove some of the systematic errors. Each virtual map cell can be labeled as one of three states: unknown, occupied, or empty. Each cell also carries a label attribute used in a re-labeling algorithm. The systematic errors result in conflicts between initial interpretations of overlapping data. These are flagged by a conflict label as new information is gathered. Cells are then re-labeled as new information is collected about the environment or if the surrounding conditions change. This approach enables us to impose simple, consistent labeling conditions to remove the conflicts once they are identified. It provides the means with which to perform straightforward data registration and mapping.

f) Alternative Path Planning

Alternative path planning is another basic component for ALX's successful navigation to its destination. It is different from that of a manipulator in that precise control is difficult for ALX because it can only have partial knowledge of its environment. The path does not have to be optimal because ALX will negotiate any given path only once while a manipulator might perform the same task thousands of times. Instead, the central issue for ALX's navigation is how quickly it can generate a safe path to the destination.

There have been a variety of proposed solutions to the path planning problem for mobile robots. Conventional path planning methods can be divided into two categories. The methods in the first category make trivial changes to the representation of a map before planning a path; the vertex graph method [25] and regular grid search method [26] fall into this category. The methods in the second category make elaborate changes to the representation of a map. The free space method [27] falls into this category. Most of the above two categories use either the graph search method or potential field approach. In the potential field approach, obstacles apply repulsive forces and the destination applies an attractive force to the robot.

Since the navigation of ALX was to be performed in real-time, and only one MC68020 single-board microprocessor was be used for both system integration and path planning, it was decided that computationally expensive path planning methods, due to either the representation conversion process or calculation of involved force fields, should be avoided in the initial development phase. Based on this consideration, a path planning scheme using a binary-tree search technique on the virtual grid map was employed initially and tested on ALX. This method is an extension of the regular grid search approach and was chosen mostly because of its simplicity compared to other methods such as the potential field method.

g) Contributors

Many people have contributed to the ALX project:

- Jon Minners set up the mobile station, provided system administration for all the computer systems and helped Micah Garlich-Miller in the vision algorithm development.
- Micah Garlich-Miller who developed the original vision processing algorithm. All the experimental results presented in this paper were obtained using Micah Garlich-Miller's algorithm.
- Kurt Kvistberg developed the software drivers to operate the motors and the range sensors.
- Curt Olson provided many further enhancements to the simulation software.

In the remainder of this report, Section 2 provides a brief description of the various subsystems on ALX, the real-time control architecture, and a computer simulation. Section 3 discusses path tracking algorithms for ALX, including those for lateral control and visual sensing. Section 4 discusses obstacle detection and collision avoidance algorithms. Simulation and outdoor experimental results are presented in Section 5 and Section 6, respectively. Section 7 gives an overall system performance evaluation of ALX and provides recommendations to the next research phase. Details on position estimation algorithms, camera calibration, and description of computer programs can be found in the appendices.

CHAPTER 2
SYSTEM DESIGN

In this section we describe ALX hardware devices and the embedded real-time computer control architecture.

2.1 System Overview

In the Robotics Laboratory at the University of Minnesota a golf cart (Figure 2-1(a)) originally automated at Alliant Techsystems [28] was used as a platform of ALX. The unit was donated to the University several years ago. Substantial modifications and redesign of the hardware system on the golf cart have been made to meet the ALX project research requirements.

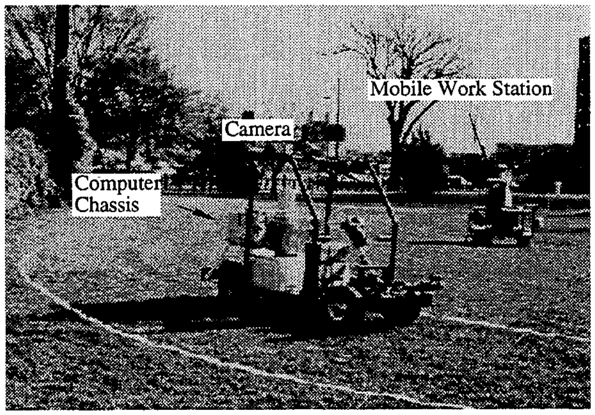


Figure 2-1 (a). ALX - Autonomous Land Experimental Vehicle

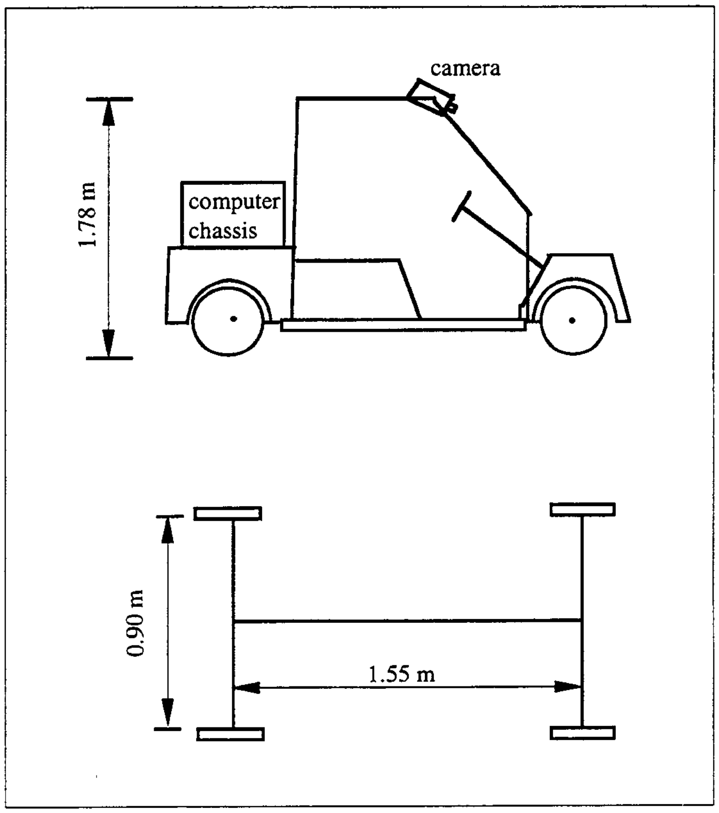


Figure 2-1 (b). ALX dimensions.

Consistent with autonomous land vehicles, ALX has a very complicated hardware design which consists of several subsystems as shown in Figure 2-2. Some of these subsystems are described briefly below.

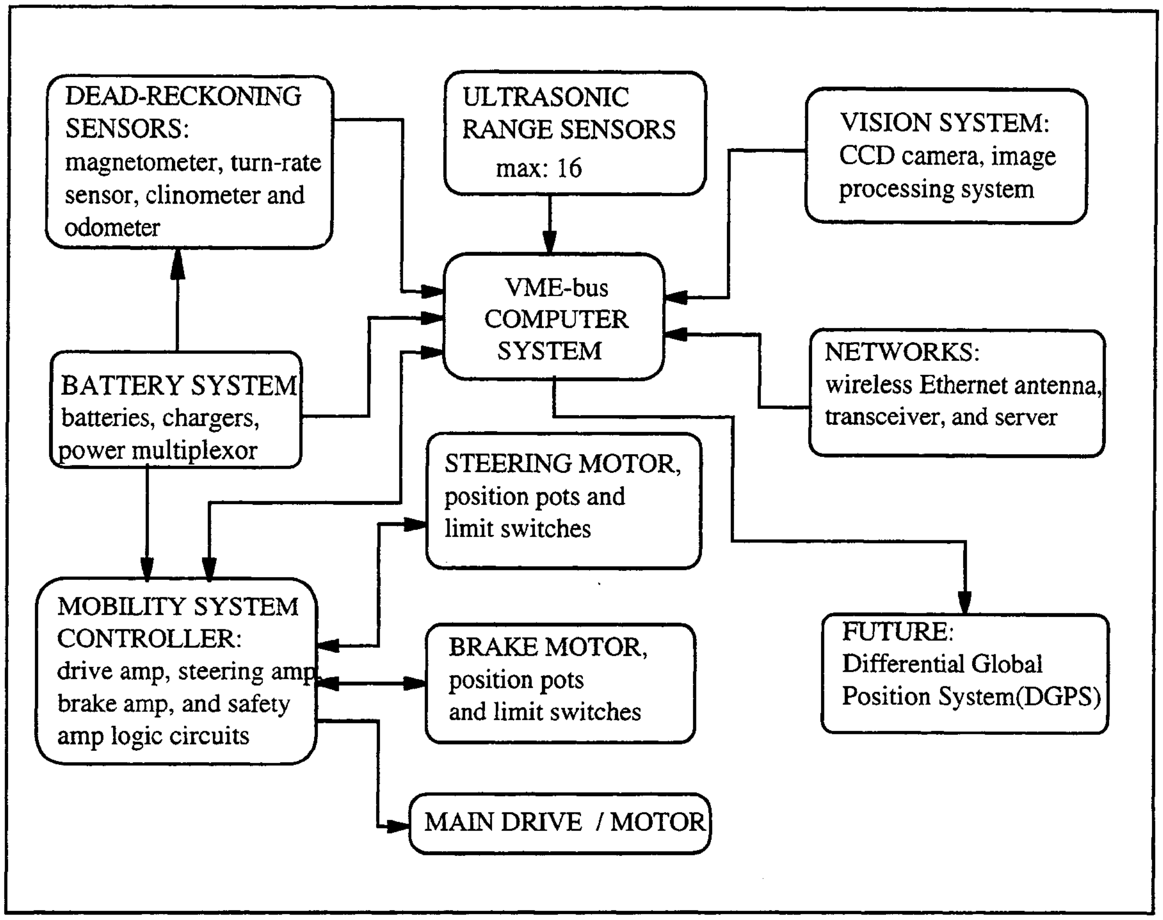

DEAD-RECKONING SENSORS: magnetometer, turn-rate sensor, clinometer and odometer
ULTRASONIC RANGE SENSORS max: 16
VISION SYSTEM: CCD camera, image processing system
VME-bus COMPUTER SYSTEM
BATTERY SYSTEM batteries, chargers, power multiplexor
NETWORKS: wireless Ethernet antenna, transceiver, and server
STEERING MOTOR, position pots and limit switches
MOBILITY SYSTEM CONTROLLER: drive amp, steering amp brake amp, and safety amp logic circuits
BRAKE MOTOR, position pots and limit switches
FUTURE: Differential Global Position System(DGPS)
MAIN DRIVE / MOTOR

Figure 2-2. System device overview

a). Dead-Reckoning System

The dead-reckoning system consists of a set of dead-reckoning sensors, which include one two-axis fluxgate magnetometer, one angular rate sensor, two clinometers, and one odometer (rear wheel encoder). The magnetometer, clinometers, and angular rate sensor are co-located in a single package (the DRS head).

The dead-reckoning system predicts the vehicle's position in Cartesian coordinate space based on position and velocity measurements from the suite of dead-reckoning sensors. Other entities sensed by ALX (roadway edges, obstacles, etc.) are then referenced to this Cartesian coordinate frame by using ALX's current computed position and orientation, i.e., the results of the dead-reckoning computation.

The DRS head is rigidly coupled to the ALX "roll cage." The wheel encoder is connected to the right rear drive wheel.

A detailed description of the algorithm used for dead-reckoning calculation is to be found in Appendix A. Because the magnetic field varies for different locales, an automatic calibration method for the magnetometer has also been developed. Details of this calibration method are also provided in Appendix A.

As with any other dead-reckoning systems, the certainty of ALX's coordinate accuracy decays as the traveled distance increases. This is due partially to slippage between the wheels and the road and to errors in the orientation estimate by the magnetometer. The methods we developed have demonstrated that successful operation is still possible despite the accumulated error of such systems. A system for continuously correcting dead-reckoning results can improve the situation. A DGPS is being tested for the same truck and may be used later on ALX to correct position estimates.

b). Ultrasonic Range Sensing System

The ultrasonic range sensor system is used to detect obstacles and to allow ALX compute and execute a trajectory around these obstacles. The information collected is accumulated and updated to construct a dynamic virtual map which provides information on the local environment to ALX. The range sensors are arranged on to detect obstacles in front and in back, as shown in Figure 2-3. The effective detection range of each sensor is a function of its mounting height, aiming angle, and its free space maximum range. Polaroid ultrasonic sensors are also affected by the weather [1]. The beam width of each sensor is about 15 degrees. The current design polls sensors at 10 Hz within each sensor group (1 to 16 sensors may be in each group). Only one sensor group is fired at a time. The firing sequence for the sensor groups is programmed in order to avoid cross-talk between sensors.

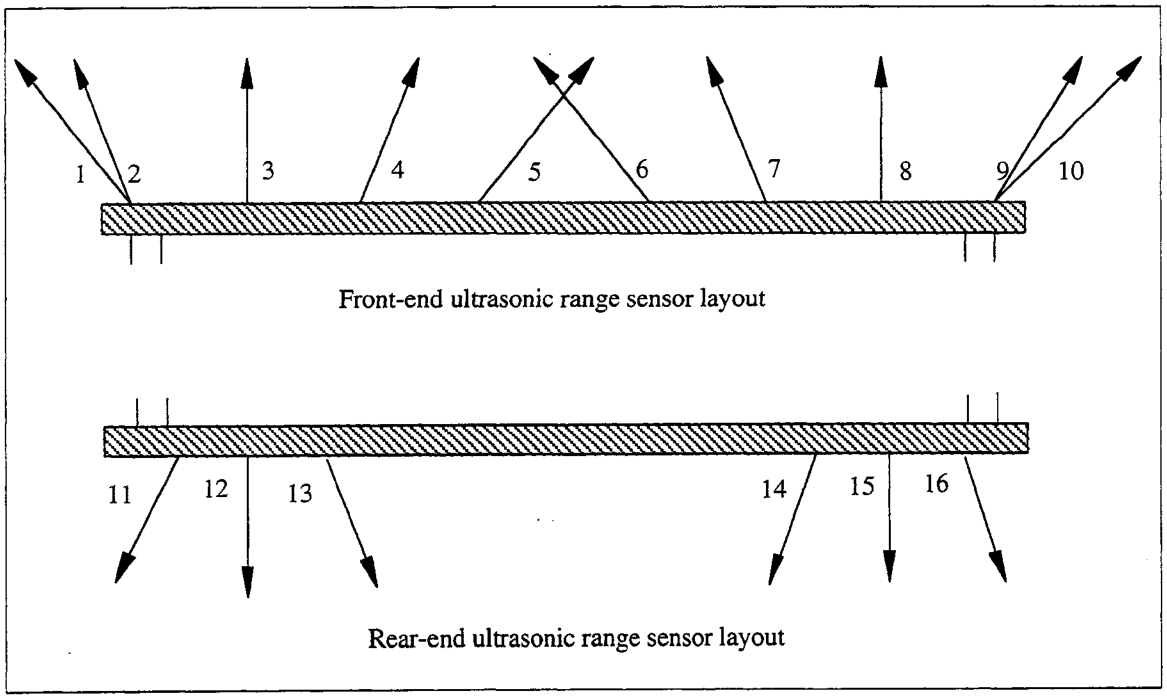

1
2
3
4
5
6
7
8
9
10
Front-end ultrasonic range sensor layout
11
12
13
14
15
16
Rear-end ultrasonic range sensor layout

Figure 2-3. Ultrasonic range sensor layout.

Figure 2-4 is a schematic drawing of the ultrasonic range sensing system. This system was designed by Don Krantz [28]. Each ultrasonic range sensor assembly is based on a Polaroid experimenter's kit. This kit includes two Polaroid 604142 transducers, two signal processing/control circuit boards, and two flextape connectors. The signal processing/control circuit board is a revision of the Texas Instruments SN28827 ultrasonic range ranging module. Each ultrasonic range sensor assembly requires a two-bit interface with the MC68020. One bit,

[1] For example, we have found that a strong head-wind will substantially decrease the detection range of a Polaroid ultrasonic sensor.

INIT, commands the sensor to begin a ranging cycle. The other bit, ECHO, signals to the HPC-46003 that an echo has been detected.

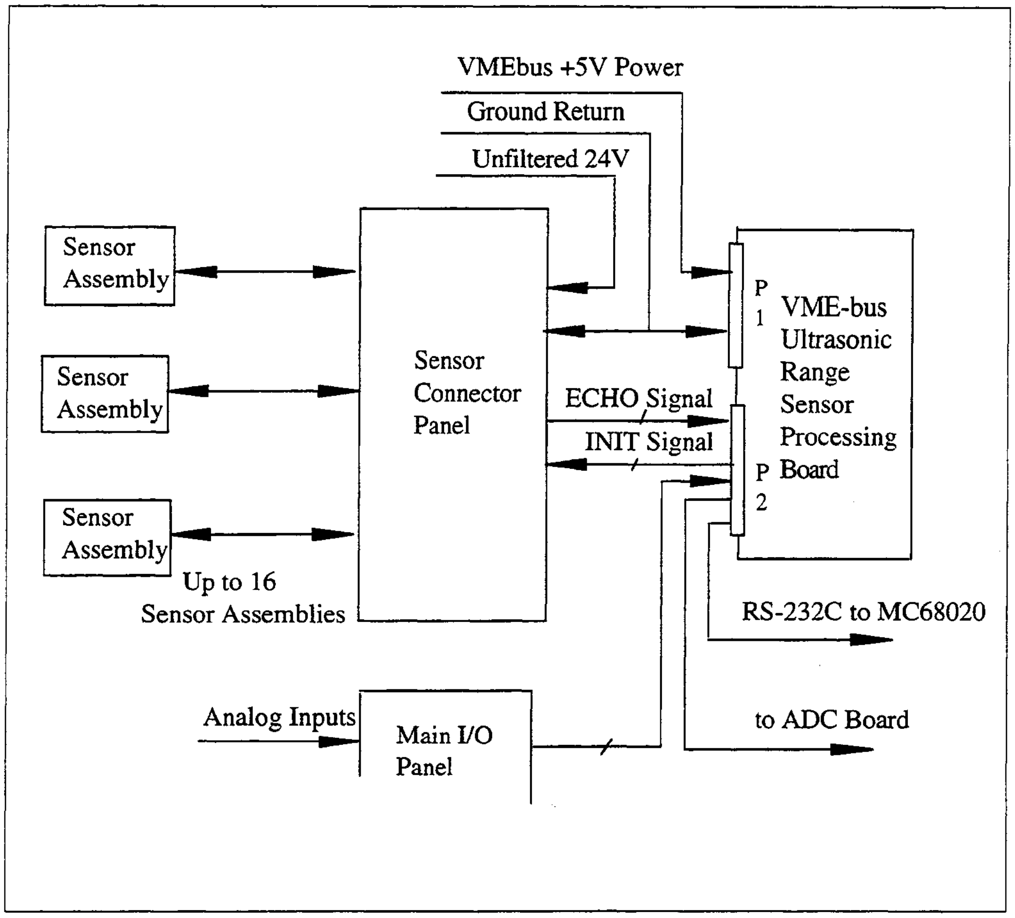


Figure 2-4. Ultrasonic range sensing system.

A VME-bus ultrasonic range sensor processing board was also designed by Don Krantz. The function of the sensor processing board is to coordinate the firing of the sensors to avoid cross-talk, measure the time-of-flight of the ranging echo pulse, and report range information to the main processor via an RS 232 serial link which operates at 9600 baud. The ultrasonic range processing board is based on the National Semiconductor HPC-46003 16-bit integrated microcontroller, which has precision timing capabilities. The CPU on board uses an 8-bit address latch and an 8-bit EPROM for program storage and the program RAM is internal to the CPU.

c). Vision System

The Vision System captures an image plus determines which features represent lane demarcation lines. The line information contained in the image is then converted to vehicle coordinate information which can be used by ALX for lateral and longitudinal control.

A 12V DC 60Hz NTSC *JAVELIN* camera is used for the vision system. It is a color MOS solid state camera with high resolution (500 TV lines). The lens chosen for this camera is a 6.5mm TV lens which gives a 55 degree horizontal angle of view. The calibration of the camera, i.e., how to map the pixel coordinates in an image to world coordinates, is discussed in Appendix C.

The image processing unit for the vision system is based on a DataCube MaxTD 200 image processor which incorporates a pipeline-processing capability (MaxVideo 200). The communication link between the image processor and the main control computer (a MVME 133 XT) on ALX is established via Ethernet.

d). Motor Battery System

The Motor Battery System (MBS) contains six 6-volt lead-acid deep-cycle batteries used for the traction motor of ALX. It also provides power to the brake and steering motors. The six batteries are connected in series to provide the nominal power of 36 Volts .

e). Mobility System Controller

The Mobility System Controller (MSC) contains the high-power control unit for the main drive motor, the steering motor, and the brake motor. The MSC uses pulse-width modulation (PWM) to control the speed of the traction motor. The PWM frequency and duty cycle are supplied as inputs. The nominal PWM frequency is 10 KHz.

The MSC provides protection against illegal input commands (e.g., both "forward" and "reverse" commands simultaneously being active). It also provides a mechanical relay disconnect for the traction motor when not in a safe state. The disconnect must be energized to supply power to the traction motor. The MSC also detects the limit switch states of the brake and steering motors, and cuts off drive power to the appropriate motor when a limit is reached.

For more details on system hardware design, refer to [28].

2.2 Real-Time Control Architecture

In traditional *sequential control system*s actions are strictly ordered as a time sequence: the behavior of the program depends only on the effects of the individual actions and their order; the time at which an action is taken is not of consequence. On the other hand, *real-time control system*s read inputs from the plant and send control signals to the plant at times determined only by operational considerations of the plant. A real-time control system differs from a sequential control system in that the sequence of some of its actions is not determined solely by the designer but also by the events occurring in the outside world in real time [29].

Responding with actions in a pre-defined order usually will not satisfy time constraints for successful vehicle control. By using Wind River's VxWorks real-time operating system to construct our computing architecture, we were able to develop an embedded real-time control system which differs from traditional control systems in that the sequence of actions is determined by real-time events occurring in the outside world.

2.2.1 VxWorks Real-Time Operating System

The Unix operating system has proven to be an excellent system for program development and for many interactive applications. However, it does not have the necessary mechanisms and utilities to support real-time applications such as control of an autonomous robotic vehicle. In the ALX's computing architecture, Wind River's VxWorks real-time operating system is used to handle critical real-time chores, while Unix is used as a cross-development host for non-time-critical applications such as editing, compilation, and linking.

VxWorks provides a development environment which includes a run-time system, testing and debugging facilities, and Unix-compatible networking facilities. The following are some of the VxWorks real-time system features that have been used for ALX software development:

- Multitasking with preemptive priority scheduling, intertask synchronization and communications facilities.
- Network facilities. "Transparent" access to other Unix systems via Unix source-compatible sockets, remote command execution, remote login, remote procedure calls, source-level remote debugging, remote file access all using Internet protocols coupled over standard Ethernet connections.
- Performance evaluation tools. An execution timer for timing a routine or group of routines, and utilities to show CPU utilization percentage by tasks.
- Shell. A C-interpreter interface that allows interactive execution of most C language expressions, VxWorks functions, and other loaded functions, and also includes symbolic references to variables.
- Module loader and system symbol table. Dynamic loading and unloading of Unix-format object modules over the networks with run-time relocation and linking.
- Debugging Facilities. Source-level debugging, a symbolic disassembler, symbolic C-subroutine traceback, task-specific breakpoints and single-stepping, system status displays, and exception-handling to safely trap and report on interrupts and hardware exception such as bus or address errors.

2.2.2 Cross-Development Environment

The hardware for the ALX computer system (golfø) includes one Unix host system (an SGI workstation), a Motorola MVME 133XT microprocessor (the VxWorks target, golfø), and a DataCube image processor. The SGI workstation is used for software development and also contains an "*image*" of the VxWorks system, which is used to boot the MV 133XT target. A Windata wireless Ethernet is used for inter-computer communications. The cross-development setup is illustrated in Figure 2-5.

Software development for ALX takes place on the Unix host: an SGI workstation which operates off an uninterruptable power supply (UPS) and can therefore be taken outdoors. Program modules are written in C using the VxWorks library, and are compiled with a C cross-compiler and then downloaded to golfø. Selected modules can be dynamically loaded across the network for testing and debugging. The VxWorks shell program is used to interactively invoke and test individual subroutines or complete tasks. Source-level debuggers are provided and allow the application to be viewed and debugged in the original source code.

Cycles of building, downloading, and testing modules are iterated until the program is ready for real-world testing. ALX then runs independently of the host workstation.

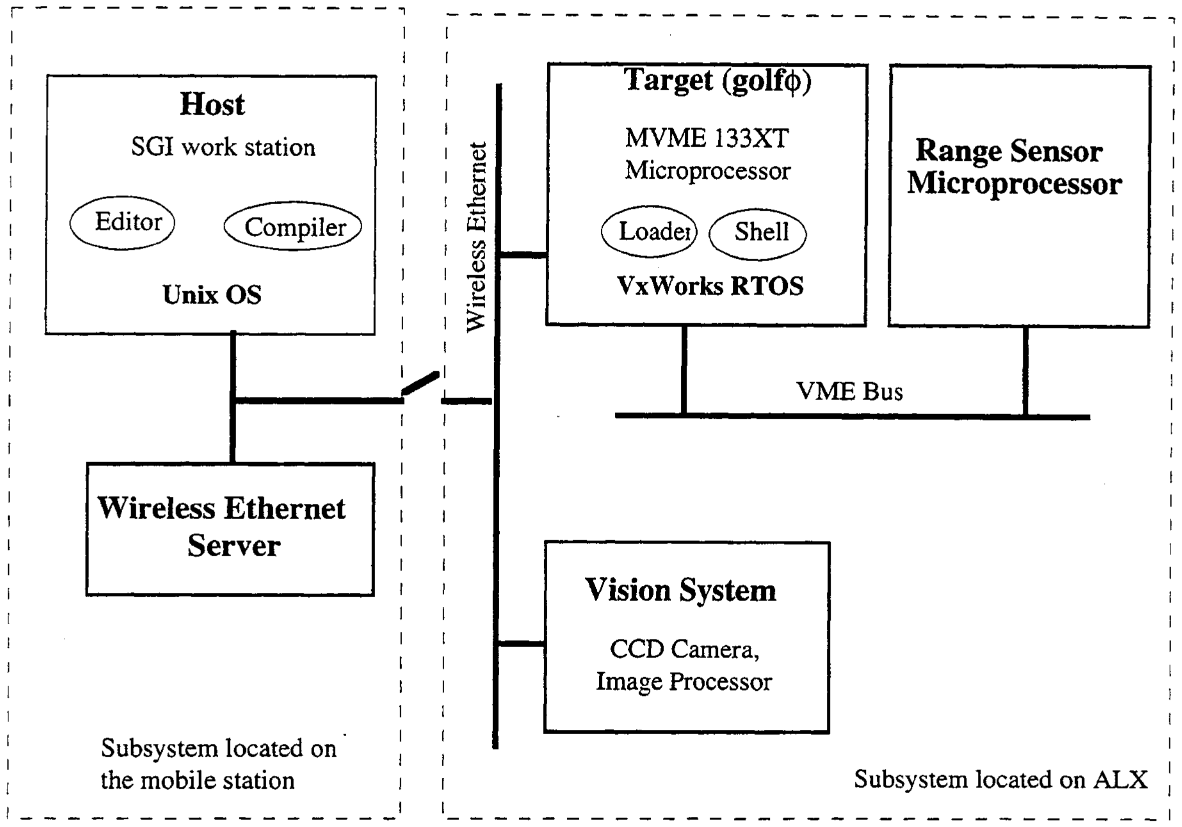


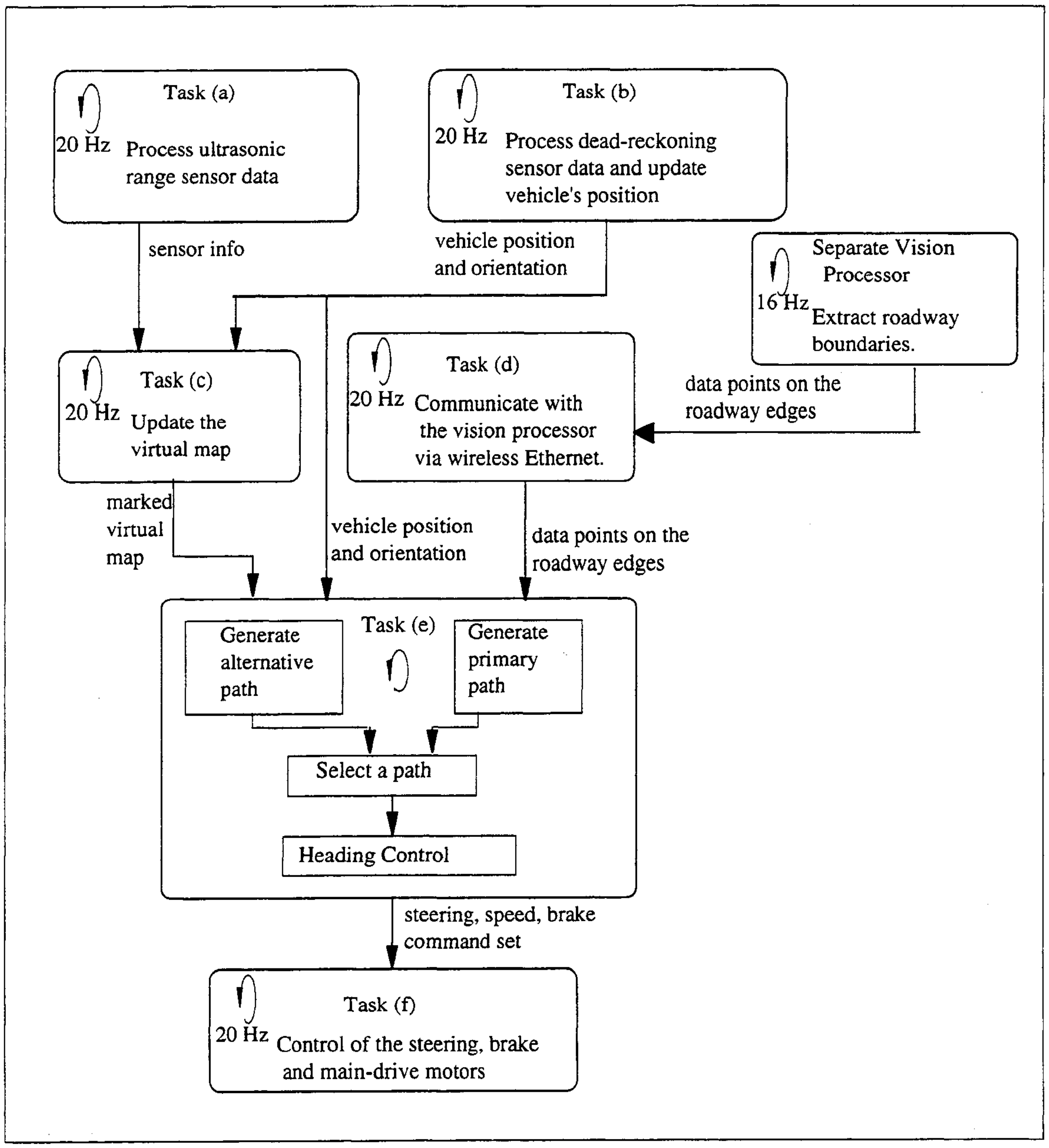
Figure 2-5. Cross development setup.

2.2.3 Multi-Tasking Control Structure

The real-time computer control system for ALX is based on the complementary concepts of *multitasking* and intertask communications.

Each apparently independent program is called a *task*. Each task has its own *context*, which is the CPU environment and system resources that the task seeks each time the kernel schedules it to run. Multitasking provides the fundamental mechanism for control and reaction to multiple, discrete real-world events. The basic multitasking environment is provided by the VxWorks real-time kernel. Multitasking creates the appearance of many programs executing concurrently when, in fact, the kernel interleaves their execution on the basis of a scheduling algorithm. In the ALX control program, there are several tasks running "concurrently," as shown in Figure 2-6.

The VxWorks multitasking kernel uses an interrupt-driven, priority-based task scheduler. With this preemptive priority-based scheduler, each task is assigned a priority and the kernel ensures that the CPU will be allocated to the highest priority task that is required to run.



Task (a)
20 Hz Process ultrasonic range sensor data
Task (b)
20 Hz Process dead-reckoning sensor data and update vehicle's position
sensor info
vehicle position and orientation
Separate Vision Processor
16 Hz Extract roadway boundaries.
Task (c)
20 Hz Update the virtual map
Task (d)
20 Hz Communicate with the vision processor via wireless Ethernet.
data points on the roadway edges
marked virtual map
vehicle position and orientation
data points on the roadway edges
Task (e)
Generate alternative path
Generate primary path
Select a path
Heading Control
steering, speed, brake command set
Task (f)
20 Hz Control of the steering, brake and main-drive motors

Figure 2-6. Multi-tasking structure.

With a preemptive priority-based scheduler, each task is assigned a priority, and the kernel ensures that the CPU will be allocated to the highest priority task that is ready to run. The scheduling is preemptive in that if a task that has higher priority than the current task becomes ready to run, the kernel will immediately save the current task's context and switch to the context of the higher priority task.

The intertask communication facilities allow multiple tasks to communicate in order to coordinate their activity. The primary intertask communication mechanism used in the MV133xt microprocessor on ALX are *message queues*. Message queues allow a number of messages, each of variable length, to be queued in a first-in-first-out order. Any task can send messages to a message queue. Any task can receive messages from a message queue. Multiple tasks can send to and receive from the same message queue.

CHAPTER 3
PATH TRACKING

We have developed a goal-point-based path tracking method which first uses image processing to extract and reconstruct roadway edges, and then uses a goal generation scheme to form a path defined by goal points, while at the same time conducting vehicle heading control by using a pure-pursuit algorithm. In this section we present the algorithms used for ALX's path tracking.

3.1 Visual Sensing of the Roadway

The goal of the vision system is to determine the location of the edges of the road in screen coordinates. To accomplish this goal, a single charge couple device (CCD) camera is used which is mounted six feet above the ground and is pointed forward and down. This camera provides an image that covers a field of view from 5 to 30 feet ahead of the cart; its average width (trapezoidal mapping) is 20 feet. The position of the lines in these images is determined using a blob analysis (or connectivity) algorithm.

3.1.1 Comparison of Blob vs Edge Detection

Initially, two vision algorithms were evaluated: blob analysis and edge detection using a Laplace of Gaussian (LoG) operator. The evaluation of these approaches was based on three main criteria: (1.) Ability to find the line in the image, (2.) Ability to eliminate or ignore noise and reflections (large bright specular regions due to the sun), and (3.) Ability to reject obstacles lying on the road that may be confused with the road edge. This evaluation was based on images consisting of white lines spray painted on green grass under varied lighting conditions. The algorithm which proved to be the most complete solution was selected.

Both blob analysis and edge detection find the edges of the road when the images are "good." The advantages of blob analysis become apparent when the images have poor lighting and exhibit specular reflections due to the sun. Noise in the image is readily eliminated by discarding the blobs based on pixel count. All blobs with less than a minimum quantity of pixels are ignored. If a reflection creates a blob greater in pixel quantity than the minimum cutoff, the blob is most likely not shaped like a line. Therefore, this reflection can be eliminated based on measures of shape (i.e. perimeter^2/area). Conversely, edge detection does not handle reflections very well. Although the LoG operator provides smoothing to eliminate noise, it is still very susceptible to noise in the images. This is because reflections of the sun on the grass often exhibit sharper contrast than the edges of the "road." When this occurs, the edges of the reflection are incorrectly considered to be lines.

Blob analysis is also superior to edge detection in rejecting the effects of obstacles on the road which appear in the image. The obstacles which ALX encounters are not shaped like a line. Therefore, blobs that are representative of obstacles are rejected as potential line data based on shape measures. Edge detection does not eliminate the obstacles as well as blob analysis. Edge detection finds the edges of the obstacles. If the obstacle is on or near the edge of the road, it is automatically considered part of the line and, as a result, skews the line data.

Due to the simplicity of the approach and the overall ability to eliminate noise and obstacles from the data, it was decided to use blob analysis for the vision algorithm. Although many of the shortcomings of edge detection can be eliminated by adding further qualifying criteria for potential lines, it was found that blob analysis provides adequate results without adding complexity to the algorithm.

It should be noted that blob analysis has one major short coming. Blob analysis is only as good as the threshold selected for the image. However, it has been found experimentally that using a histogram based thresholding method consistently reduced these image to the desired data (lines with some noise and obstacles). See appendix C.1 for more information on the thresholding method.

3.1.2 Blob Analysis

Blob analysis is a relatively straight-forward algorithm. It involves first thresholding the image (often first smoothed with a Gaussian filter). The resulting binary image is then evaluated for connectivity (connectivity analysis). In other words, neighboring pixels that are "on" or which have a value of 1 are grouped together. The resulting groups are considered blobs. The final step of blob analysis is gathering data about the blobs, for instance, their shape or size as well as their relationship to one another.

Although the blob analysis is straight-forward it is usually computationally expensive. Connectivity analysis alone has several algorithms designed to enhance computation time. Determining the relationship between blobs can also be costly.

The blob analysis algorithm discussed in this section addresses the problem of minimizing the computation time as our highest priority. Blobs relationships to one another are not calculated as this is not a requirement of this system. Information about a blob being a parent or a daughter to another blob is not required for determining the edges of the road. Also the connectivity analysis algorithm has been created to allow for efficient determination of blobs. The details of the algorithms are discussed in the following sub-sections.

3.1.2.1 Datacube Image Processing System

The vision algorithm is executed on a MaxTD pipeline based image processor. The MaxTD is a VMEbus based computer that contains a Motorola MVME 167 (68040 based) processor and MaxVideo 200 pipeline hardware. This vision algorithm uses both the MV200 pipelines and the

MVME 167 to perform the image processing (see Figure 3.1 for flow chart). Likewise the following sub-sections describe first the pipelines that are executed and then the processing performed by the Motorola processor.

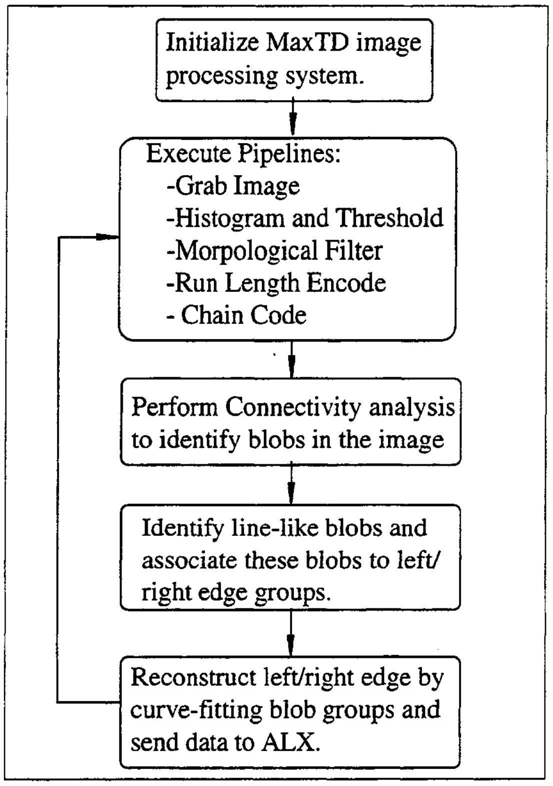

Initialize MaxTD image processing system.
Execute Pipelines:
-Grab Image
-Histogram and Threshold
-Morpological Filter
-Run Length Encode
- Chain Code
Perform Connectivity analysis to identify blobs in the image
Identify line-like blobs and associate these blobs to left/ right edge groups.
Reconstruct left/right edge by curve-fitting blob groups and send data to ALX.

Figure 3-1. Simplified flow chart of blob analysis vision algorithm.

3.1.2.2 Data Pipelines

A pipeline is a software programmable hardware path on the MaxVideo 200 that performs a desired task (e.g., convolution) on the input data. The pixel values are sent through a pipeline as a stream of data and the desired operation is performed on the image. All data pipelines are programmed using ImageFlow.

The blob analysis requires four data pipelines. The first pipeline is used to capture the image. The results are passed to second pipeline which collects histogram data and also performs a thresholding operation. Next the third pipeline performs a 3 x 3 morphological filter. The third pipeline then uses the filtered image to create a run length encoded image as well as a chain coded image. The output of the morphological filter is displayed on the screen by a fourth pipeline with lines fit to appropriate blobs (the line fitting is discusses in the section 3.1.2.4). The input/output pipelines are not discussed as they are trivial implementation but the two main processing pipelines will now be discussed.

Note that due to the complexity of the Datacube system, the pipelines are not discussed in exact detail. Persons not familiar with the Datacube and ImageFlow programming will probably not be able to follow the detail required for pipeline construction. Persons interested in the details of the pipeline can refer to the program descriptions in Appendix D, table D-2, and references [48] and [49].

The histogram and thresholding pipeline (Pipe2) is a multi-destination pipeline. The destination surfaces are oApViewSurf and oMem4InSurf (see Fig. 3.2). The oApViewSurf stores the data from the histogram operation while the oMem4InSurf stores data from the thresholding operation. The important thing to remember about this pipeline is that having two destination surfaces saves time by performing two processes simultaneously. Of course the processes can not have conflicting resource (both processes can not use the same element). The oMem4InSurf stores the thresholded image and is the input for the next pipeline. Note that although the data for the histogram has been extracted by a pipeline this data is evaluated on the MVME 167 for selecting an appropriate threshold value. In order to be time efficient, threshold selection is performed on the MVME 167 after completion of this pipeline while the next pipeline is executing. The threshold value selected by this method is then used on the following frame. Threshold selection process is covered in appendix C.1 "Tuning and Running Blob Analysis."

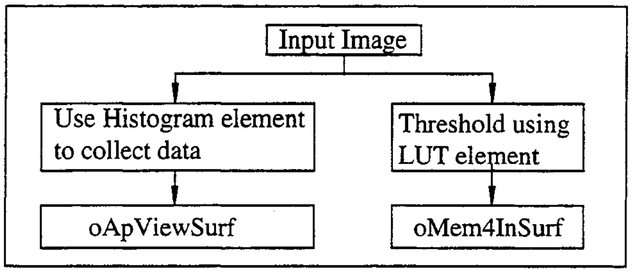

Input Image
Use Histogram element to collect data
Threshold using LUT element
oApViewSurf
oMem4InSurf

Figure 3-2. Threshold and histogram Data Pipeline.

The other pipeline (Pipe3) is also a multi-destination pipeline. The destination surfaces are oMem0InSurf, oMem1InSurf and oMem3InSurf and these surfaces store data from the morphological filter, the chain coding process and the run length encoding process respectively (refer to Figure 3-3).

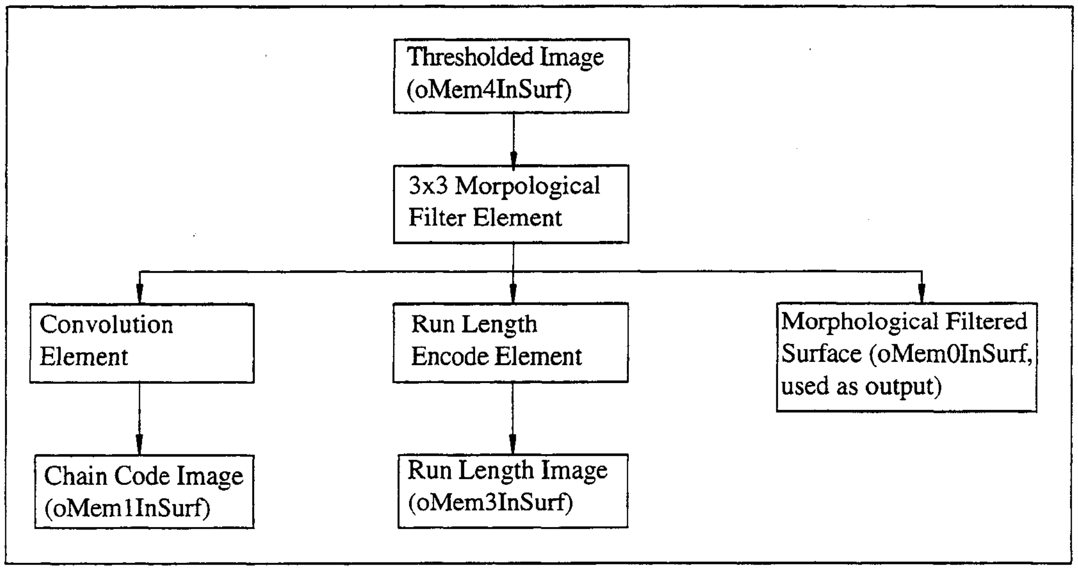

Thresholded Image (oMem4InSurf)
3x3 Morpological Filter Element
Convolution Element
Run Length Encode Element
Morphological Filtered Surface (oMem0InSurf, used as output)
Chain Code Image (oMem1InSurf)
Run Length Image (oMem3InSurf)

Figure 3-3. Morphological filter, run length encode and chain code data pipeline.

The morphological filter is used to improve the quality of the thresholded image. This filter moves across the image looking at a 3x3 pattern of pixels and replaces the center pixel based on a look up table (LUT) and the pixel patterns. This filter effectively eliminates isolated points and enhances the connectivity of blobs by filling in gaps in the connected regions. The output of the filter is passed on to the chain coding and run length encoding processes and is also stored in oMem0InSurf which is used to display the results.

0	0	0	1	1	1	1	0
0	1	0	1	1	0	0	0
0	0	1	1	1	1	0	0

A. Binary Image

1	2	3	129	130	131	132	1
1	129	1	129	130	1	2	3
1	2	129	130	131	132	1	2

B. Run Length Encoded Image

Figure 3-4. Example of a Binary Image(A) and the Corresponding Run Length Encoded Image(B).

The run length encoded image contains information on the position of 0 to 1 and 1 to 0 transitions on each line of the binary image and is used in connectivity analysis (its use is discussed in section 3.1.2.3). The run length encoding process takes as input the least significant bit (the bit representing on or off in the binary image) and provides an 8-bit output. For each pixel value of the output image, the 7 least significant bits contain a count of the pixels since the last 0 to 1 or 1 to 0 transition in the binary image (counting from left to right). The most significant bit of each pixel value of the output image is the value of the binary image. Note that on the run length encoded surface a count of 0 represents a distance of 128 times N (N being any positive integer) pixels from the last transition. Figure 3-4 shows an example binary image and the corresponding run length encoded image.

The chain code image is the same as that used in [50]. Chain code images usually contain information regarding the direction and length of a line segment of connected pixels. In this case information about connected neighbors is only recorded. This is accomplished by convolving the image with the following 3x3 kernel:

1	2	4
8	0	16
32	64	128

Applying the above kernel to a binary image results in the chain coded image having values from 0 to 255 depending on the values of the connected neighbors.

At this point the data pipelines used by this algorithm have been defined, but how are they executed? The input and output pipelines are initiated and execute continuously but Pipe2 and Pipe3 are cannot be continuous due to conflicts in resources. These pipes are programmed such that Pipe3 begins whenever Pipe2 completes. Pipe2 begins when a dummy event is triggered. Therefore, a triggering of the dummy event causes both Pipe2 and Pipe 3 to execute in that order. To allow for efficient operation the dummy event is triggered after the connectivity analysis has extracted the required data from the run length encode and chain coded images. Figure 3-5 shows the flow chart of the vision algorithm with the pipelines executing in this manner. This allows the pipes to execute while the MVME 167 is processing other data. Running the processes in parallel in this fashion accounts for approximately a twenty-five percent reduction in cycle time.

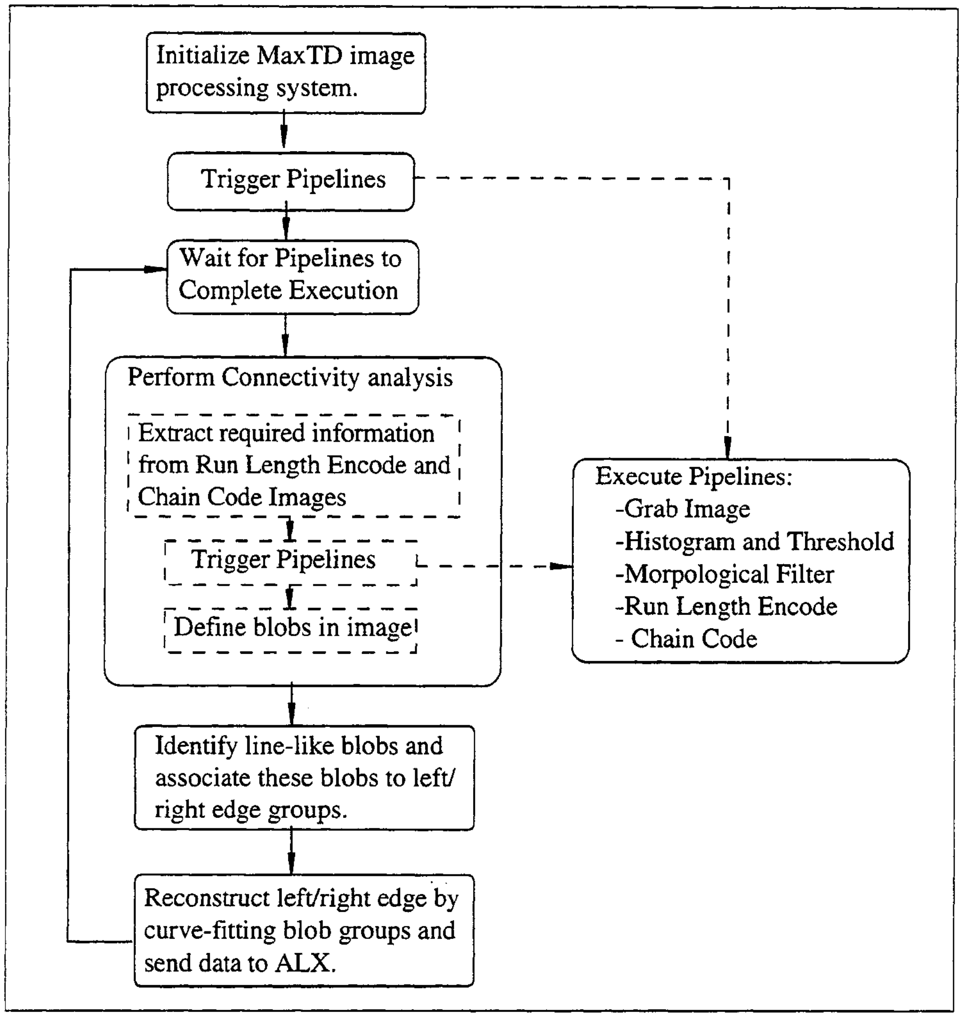

Initialize MaxTD image processing system.
Trigger Pipelines
Wait for Pipelines to Complete Execution
Perform Connectivity analysis
Extract required information from Run Length Encode and Chain Code Images
Trigger Pipelines
Define blobs in image
Execute Pipelines:
-Grab Image
-Histogram and Threshold
-Morpological Filter
-Run Length Encode
- Chain Code
Identify line-like blobs and associate these blobs to left/ right edge groups.
Reconstruct left/right edge by curve-fitting blob groups and send data to ALX.

Figure 3-5. Flow chart for vision algorithm - modified.

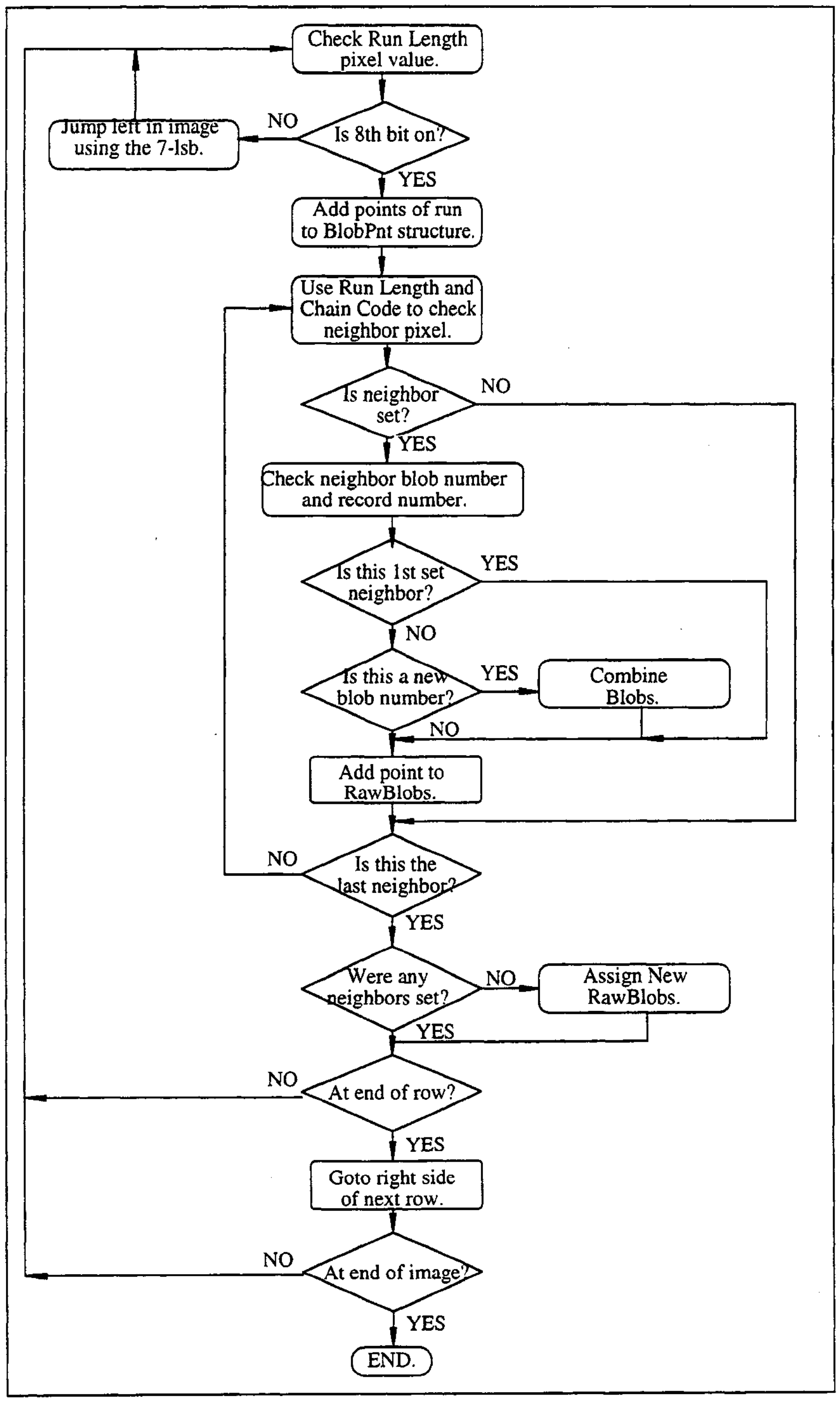
3.1.2.3 Connectivity Analysis

Once the run length encode and the chain code image have been generated by the Datacube MaxVideo 200 the next step is to perform the connectivity analysis. The connectivity analysis involves extracting desired information for each blob from the processed images. This process could also be implemented via Datacube pipelines, however the programming process would be tedious. Since the data to process has already been greatly reduced, a less involved programming task, which is also time efficient, can be implemented on the MVME 167 board. The connectivity analysis and the remainder of the program are executed on the MVME 167.

To extract the connectivity information, two data structures where created. The first is for a point in a blob, BlobPnt. The BlobPnt structure is a statically allocated array and contains information on the screen position of the blob point as well as the index (for the BlobPnt structure) of the next point in the blob. The second structure is for the segments of blobs, RawBlobs. RawBlobs is also a statically allocated structure and contains the indices of the beginning and last point of the blob in the BlobPnt structure. It also contains the pixel count for the blob.

The above structures do not store the blob number for each pixel. The blob number that each pixel is a part of is stored in that pixel's memory location in the run length and chain code images (the msb of the blob number is stored in the run length image and the lsb of the blob number is stored in the chain code image allowing for up to 65536 blobs). This approach allows for checking which blob a pixel is a member of quickly. The run length and chain code image are already being traversed using pointers. Checking a pixel's blob number involves incrementing the pointers and performing a logical OR. More importantly this memory mechanism allows checking what blob a pixel belongs to without accessing the BlobPnt structure. This reduces the complexity of the connectivity analysis as well as the complexity of the BlobPnt data structure.

Data for the RawBlobs and BlobPnt structures is collected using the run length encoded and chain coded images (refer to Figure 3-6 for flow chart). The run length encoded image is used to locate all the pixels that are set in the image and the chain coded image is used to determine connectivity at these points. The image is traversed right-to-left and top-to-bottom by jumping only to set pixels using the information from the run length encoding. The chain code and the run length image are now used to determine if the neighbors above or to the right of the run of set pixels are also set (pixels below or to the left have not been visited yet). If there are no set neighbors, all of the points in the run are added to the BlobPnt structure and a new RawBlobs is added. The number of the new blob is now stored in the run length and chain coded image in the location of the set pixels for that blob. If there are set neighbors, the blob number of the RawBlobs is determined from the values at that pixel location of the run length and chain coded images. The run of set pixels is then added onto the appropriate RawBlobs structure and the blob number is stored for each run point. When one run is connected to more than one RawBlobs, all of the blobs are connected into one blob.



Check Run Length pixel value.
Is 8th bit on?
NO
Jump left in image using the 7-lsb.
YES
Add points of run to BlobPnt structure.
Use Run Length and Chain Code to check neighbor pixel.
Is neighbor set?
NO
YES
Check neighbor blob number and record number.
Is this 1st set neighbor?
YES
NO
Is this a new blob number?
YES
Combine Blobs.
NO
Add point to RawBlobs.
Is this the last neighbor?
NO
YES
Were any neighbors set?
NO
Assign New RawBlobs.
YES
At end of row?
NO
YES
Goto right side of next row.
At end of image?
NO
YES
END.

Figure 3-6. Connectivity analysis flow chart.

3.1.2.4 Blob Association and Curve Fitting

After the image has been completely traversed and the information is collected for the RawBlobs, the next step is to determine which blobs have acceptable line shapes and to save these blobs. There are two criteria that are used for this evaluation: blob area and blob elongation (perimeter^2/area). Blobs that meet the minimum area and elongation requirements are saved in the GoodBlobs data structure. The data in GoodBlobs are considered as potential candidates for part of the road lines. These blobs are associated with the left-side edge, the right-side edge or are discarded; this process is called blob association.

To carry out this function correctly, a memory mechanism was used. The program keeps track of the road edges reconstructed in the previous image frame and then uses them as references for the current image's blob association. This mechanism is based on the assumption that the image processing speed is reasonable in terms of ALX's operating speed so that the positions of the road edges in two consecutive images do not differ significantly. As shown in Figure 3-7, with the previous edges (dashed lines) as references, some of the line-like blobs are associated together to represent either the left or right roadway edge; and some line-like blobs (for example, blob *"h"*, *"j"* and *"i"*) are discarded because they cannot be associated with either the previous left edge or the previous right edge.

After the blob association, a least-square curve fitting approach is used to construct a second order polynomial curve which represents each of the two roadway edges. To represent the constructed road edges, ten evenly-spaced horizontal lines are scanned across the image, and the intersection points between the horizontal lines (dashed in Fig. 3-8) and the left (or right) edges are used to represent the left (or right) edge (Fig. 3-8). These intersection points are then transferred to the ALX computer for processing the road data. The results of the morphological filter and of the blob analysis and final road boundaries are displayed by the output pipeline on the MaxVideo 200.

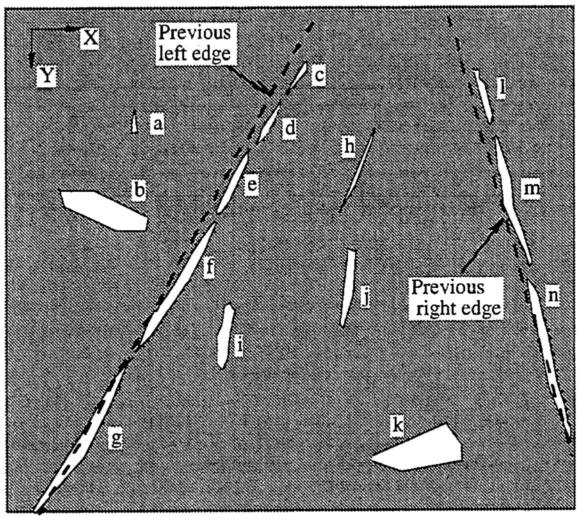


Figure 3-7. Binary image of the roadway with previous image lines.

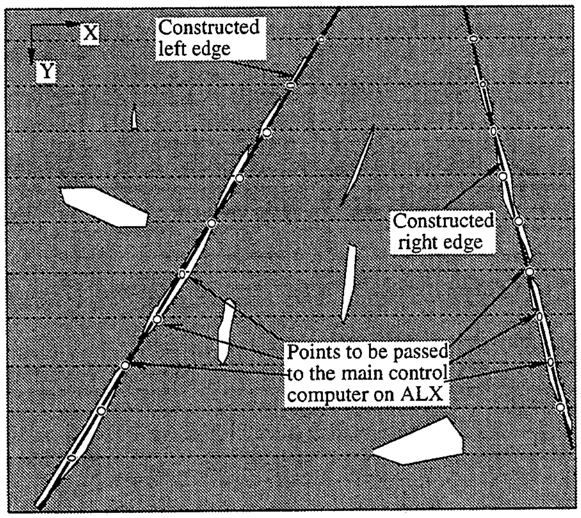


Figure 3-8. Reconstructed roadway edges.

3.2 Path Generation

In this section, we present the algorithm that ALX uses to define primary forward path based on *goal* points. These goal points are generated from the roadway image information passed by the vision system .

3.2.1 Path Based on Goals

In our path tracking algorithm, any path, either generated from the visual processing unit or from the alternative path planning algorithm when the vehicle is obstructed (see Section 4 for detail), is represented by a finite number of points “marking” the center-line of this path. These points are called *goals*. Goals are defined in world coordinates, and represent the positions which the vehicle is directed to reach. Information for goals, i.e., their (x,y) coordinates, are stored in a data structure called *goal stacks*. A stack is a linear data list which operates in a first-in-first-out manner, i.e., the manipulation of its components takes place only at the beginning of the list, and the last component put into the stack remains at the top of the stack for the next immediate operation. There are two operations on a goal stack, namely “push” and “pop.” The push operation inserts a goal point onto a goal stack, while the pop operation “pulls" the first top goal point in the goal stack, i.e., it removes the information for this goal from the goal stack and returns it to the program. These operations are illustrated in Figure 3-9 (a) and (b).

Figure 3-9 (a) shows an example of a path, its corresponding goal stack, and the operation of the stack during ALX's path tracking. Several points which are on the center line of the path are selected as goal points and are "pushed" into the goal stack. The goal closest to the vehicle is at the top of the goal stack, and thus it is ready to be immediately

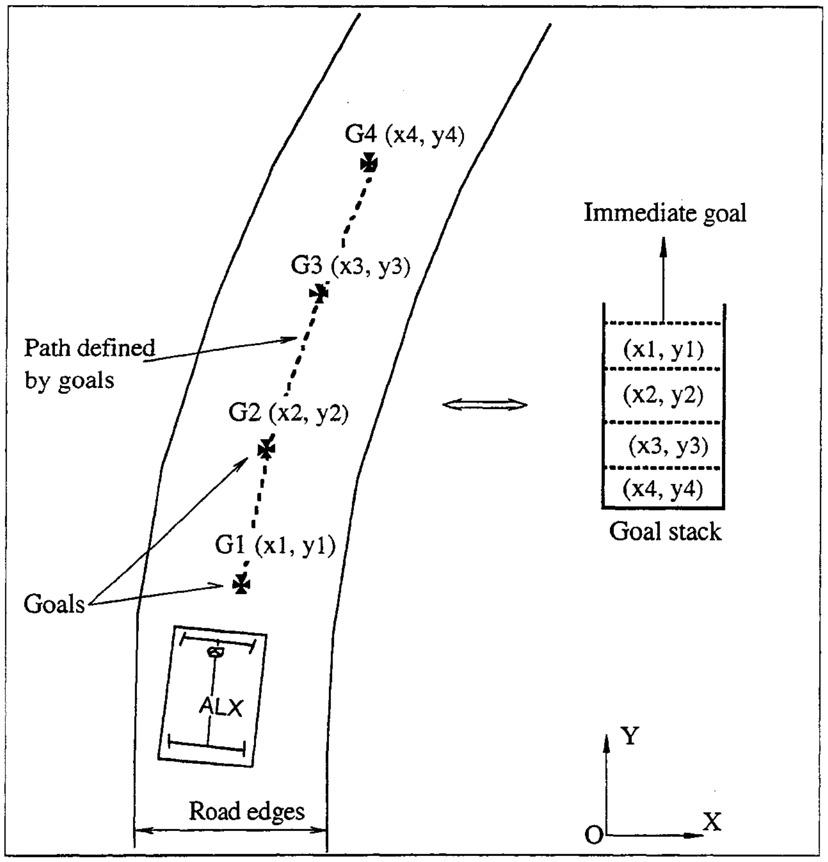


Figure 3-9 (a). Illustration of the goal stack concept and operation.

Goals are generated and stored in the goal stack.

referenced to generate steering commands. The goal farthest away from the vehicle is pushed to the bottom of the goal stack and will be used only after the vehicle has successfully reached all the positions indicated by previous goals (Figure 3-9 (b)).

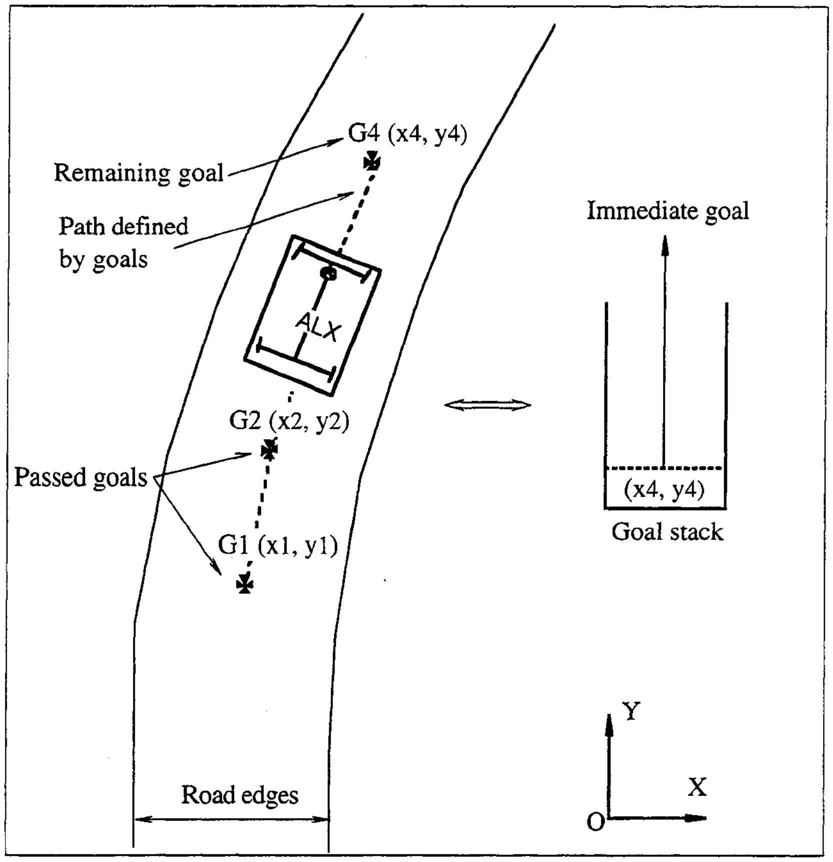


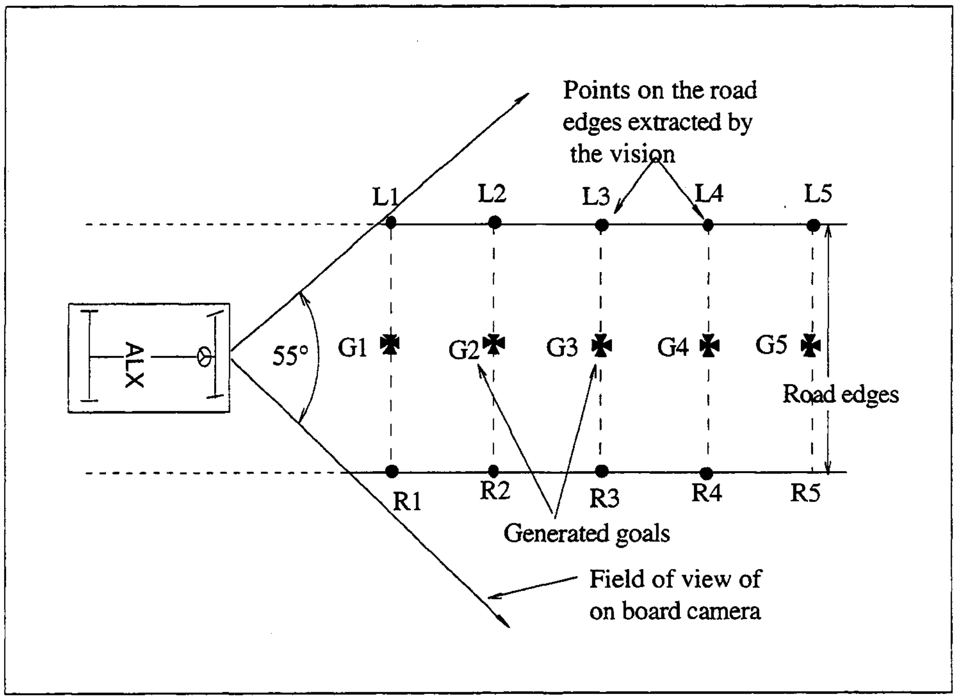
Figure 3-9 (b). Illustration of the goal stack concept and operation.

ALX reaches each of the goals during its operation.

3.2.2 Goal Generation

In Section 3.1, we described how the roadway edges (the white lane marks) are extracted during the image processing. For each roadway edge, a polynomial curve is constructed to represent this edge. A set of points evenly apart in vertical direction on the polynomial curve is selected and then is transmitted by the vision system via Ethernet communication to the MC68020 main control microprocessor on ALX. These points indicate the positions of the white lane marks in the image. After this transmission, the coordinates of these image pixel points are first transformed into local coordinate frame with respect to the mounting position of the camera, and then, by using the dead-reckoning information, they are further transformed to the world coordinate frame which is set up at the time when ALX conducts dead-reckoning initialization. The coordinate transformation details can be found in Appendix B.

Next, we describe the algorithm for generating goals from these transmitted points that are on the roadway edge(s).



Points on the road edges extracted by the vision
L1
L2
L3
L4
L5
ALX
55°
G1
G2
G3
G4
G5
Road edges
R1
R2
R3
R4
R5
Generated goals
Field of view of on board camera

Figure 3-10 (a). Goals are generated from two evenly visible roadway edges.

Under all but the most degenerate situations, ALX will find itself in one of two vision situations. The first is the most ideal. In this case, the on-board camera views the center of the roadway evenly to the road edges. The vision system is able to recognize both road edges, and the same number of points on both of the edges is selected. For example, in Figure 3-10 (a), points *L1, L2, ..., L5* are selected to represent the left edge of the road, while points *R1, R2,..., R5* are selected to represent the right edge. Goal *G1* is the goal generated closest to ALX and is the center point of line segment *L1R1*. Goals *G2, G3, G4* and *G5* are generated in the same manner.

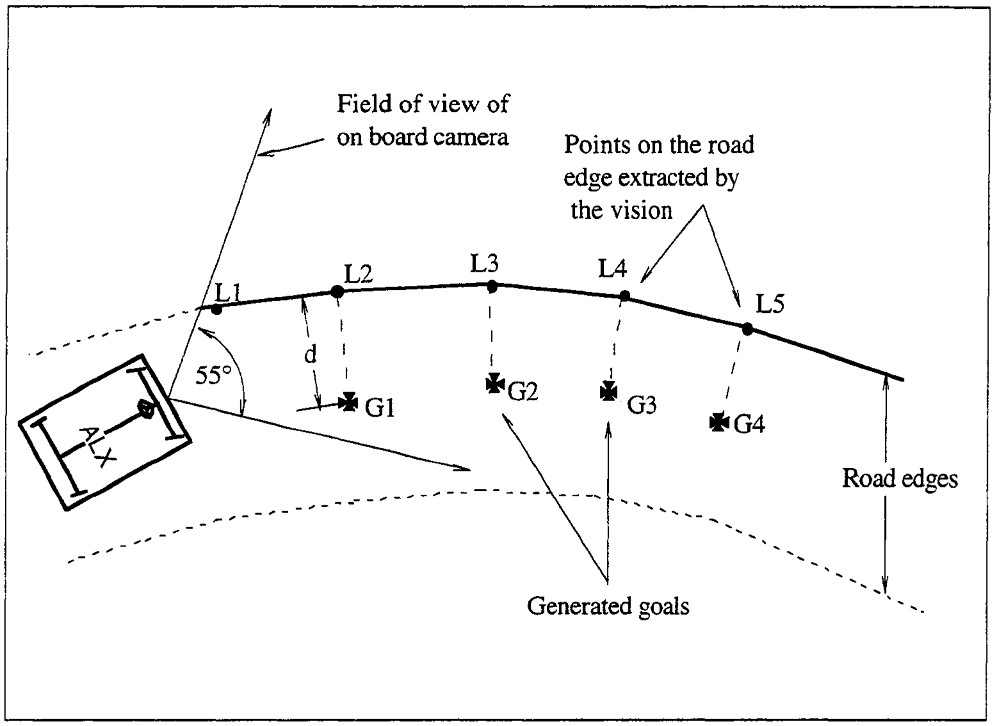

Field of view of on board camera
Points on the road edge extracted by the vision
L1
L2
L3
L4
L5
55°
d
G1
G2
G3
G4
ALX
Road edges
Generated goals

Figure 3-10 (b). Goals are generated from only one visible roadway edge.

Often when avoiding an obstacle or negotiating a turn, ALX enters a state where one of the road edges is obscurred; Figure 3-10 (b) is an example of such a case. Only points *L1, L2, ..., L5* on the left edge can be identified and passed to the main computer. When this occurs, the previous method of line segment division can no longer work and the method for calculating goal points from this limited vision data is as follows. Since point *L1* is the closest point to ALX on the left edge, point *L2* is the next selected point. The algorithm calculates a normal to the line segment *L1L2* at point *L2*. A goal is placed on this normal at a predefined distance *d* towards the center of the road. Thus, the goal *G1* is located near the center of the road, and is the goal closest to ALX. Goals *G2*, *G3*, and *G4* are generated in the same manner.

We assume the width of the road on which ALX operates to be relative constant. In this case, distance *d* is predefined to be slightly in excess of half of the road width.

Figure 3-10 (c) illustrates a more general case than the first two discussed above: less of one of the road edges is visible than of the other one, but it is still partially in the view of the camera. In this situation, the method to generate goals is a hybrid of those used in the first two extreme cases.

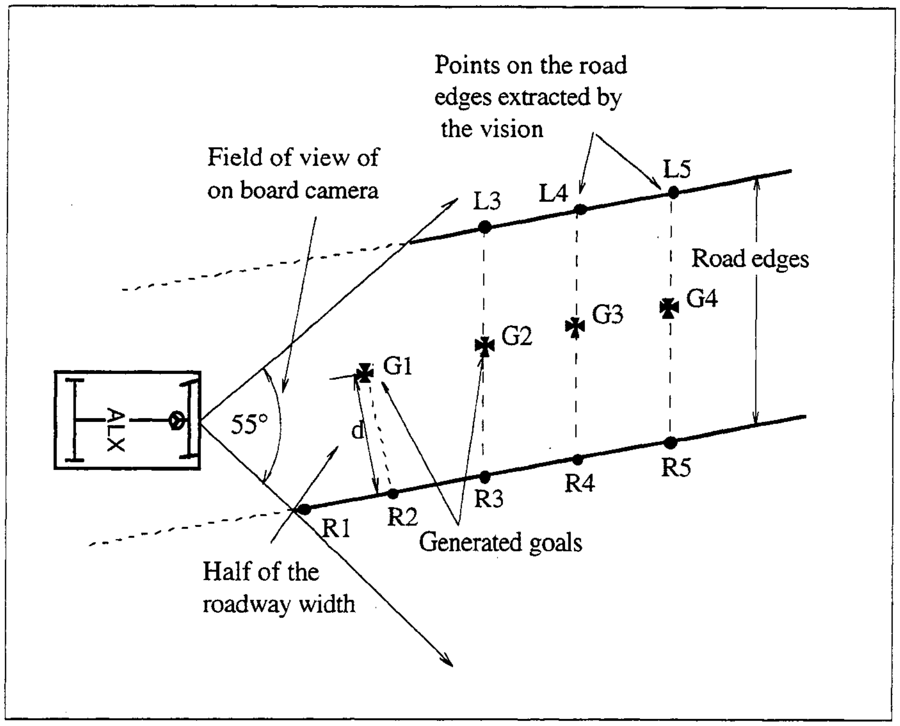

Points on the road edges extracted by the vision
Field of view of on board camera
L3
L4
L5
Road edges
G1
G2
G3
G4
ALX
55°
d
R1
R2
R3
R4
R5
Generated goals
Half of the roadway width

Figure 3-10 (c). Goals are generated from two unevenly visible roadway edges.

3.3 Heading Control Scheme

ALX is developed on the chassis of an Yamaha electric golf cart, which is essentially an Ackerman-steered, non-holonomic vehicle. For this kind of vehicle, it is not possible to achieve an arbitrary trajectory, but it is possible to achieve an arbitrary position, using multiple approach maneuvers [28]. It is also possible to follow paths within certain constraints (for example, any three goal points must define a circle with a radius larger than the vehicle's turning radius). Our approach is to develop a vehicle heading control scheme and supporting vehicle inverse kinematics equations.

Our vehicle heading control scheme is a variation of the "pure-pursuit" method, proposed by Amidi [31] and implemented by Murphy [32] at Carnegie Mellon University. This method uses data points (called target points) from a known distance (Murphy used 6-12 meters; we empirically determined a distance about twice the length of the wheelbase) ahead of the vehicle to compute a steering command. A new point is chosen each control cycle and the vehicle will drive towards that target point. This method has been proven to be simple and effective, and Murphy reported smooth control at speeds in excess of 25 MPH [32]. It was later also implemented by NIST for its demonstration of autonomous road following [33].

Our algorithm is loosely based on Murphy's "pure-pursuit" concept, with new control equations. The geometric relationships are shown in Figure 3-11.

The first step of the algorithm is to select a controlled point *CP* on the vehicle. If moving forward, the controlled point is chosen to be the center point between the front wheels. If moving in reverse, the controlled point is the center point between the rear wheels.

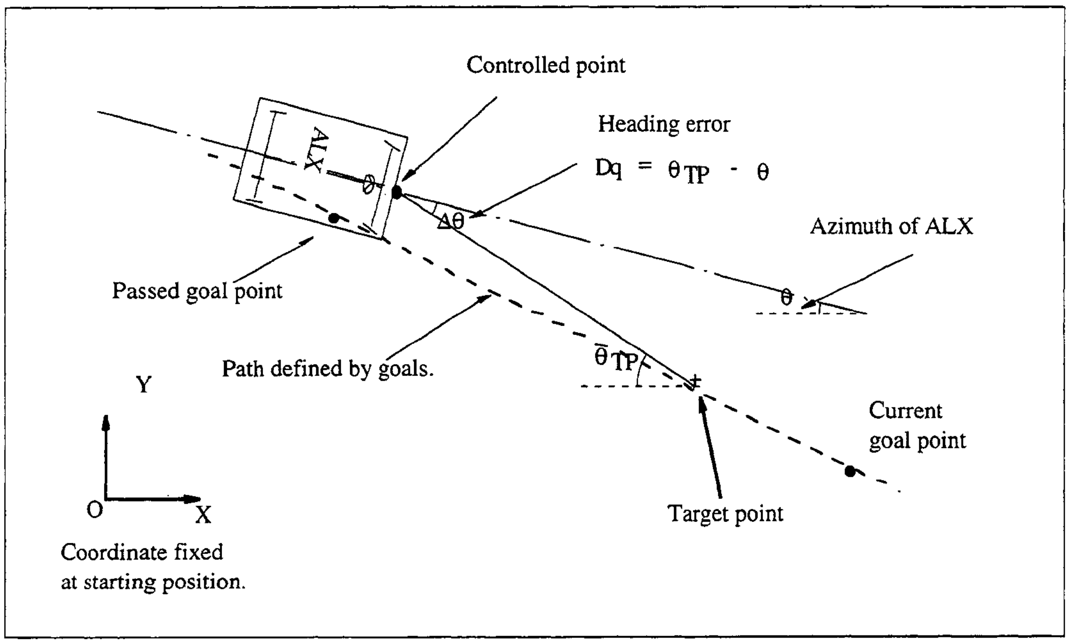


Figure 3-11. Heading control of ALX.

The next step is to determine whether ALX has moved "past" the current path segment, defined by goal points *Ga* (x_a, y_a) and *Ga* (x_b, y_b). This is done by finding the normal from the controlled point to the line containing the current path segment *GaGb*, and by finding the coordinates (x_i, y_i) of the intercept of the normal with the line containing the path segment *GaGb*. The slope of the path segment *GaGb* is:

$$m = tan(\theta) \tag{3-1}$$

The slope of the normal to the path segment *GaGb* is:

$$m_n = -\frac{1}{m} \tag{3-2}$$

The coordinates of the controlled *CP* point to be (x_{cp}, y_{cp}), and the coordinates of the interception point *I* are:

$$x_i = \frac{(m_n x_{cp} - y_{cp}) - (m x_a - y_a)}{m_n - m} \tag{3-3}$$

$$y_i = m(x_i - y_{cp}) + y_{cp} \tag{3-4}$$

Next, we determine if the intercept is contained within the path segment *GaGb*. If point *Ga* is defined to be the "first" point, i.e., the point along the path closest to the origin of the world coordinate frame, then the intercept position can be determined to be "*before*", "*within*", or "*after*" the path segment using the following Boolean tests:

$$\text{“before” } \mathit{iff}\ dist(I, Ga) + dist(Ga, Gb) = dist(I, Gb) \tag{3-5}$$

$$\text{“within” } \mathit{iff}\ dist(I, Ga) + dist(I, Gb) = dist(Ga, Gb) \tag{3-6}$$

$$\text{“after” } \mathit{iff}\ dist(I, Gb) + dist(Ga, Gb) = dist(I, Ga) \tag{3-7}$$

where the "*dist*" function is the Euclidean distance between two points, and "*iff*" means the statement of "if and only if". If the intercept is determined to be "*after*" the path segment, then the next path segment becomes the current path segment, and the test is repeated.

The next step is to select a target point *TP* to steer towards. *TP* is selected by moving along the path by a distance δ and jumping to the next segment if necessary. If δ is too small, the vehicle will "jitter", especially at path segment transitions. If δ is too large, the vehicle will "cut" corners. A good compromise for δ determined empirically is to make it approximately equal to the wheelbase of ALX.

Refer to Figure 3-11. Once a target point is selected, the heading error angle θ_{TP} to the target point *TP* from the controlled point *CP* can be determined:

$$\theta_{TP} = atan2(y_{TP} - y_{CP}, x_{TP} - x_{CP}) \qquad (3\text{-}8)$$

$\Delta\theta$ is then the difference between the vehicle's azimuth θ and θ_{TP}, as shown in Figure 3-5. A slight change is needed if the vehicle is traveling in reverse and θ is defined with respect to the front of the vehicle:

$$\Delta\theta = \theta_{TP} - \theta \qquad \text{(forward)} \qquad (3\text{-}9)$$

$$\Delta\theta = \theta_{TP} - (\theta + \pi) \qquad \text{(reverse)} \qquad (3\text{-}9)$$

$\Delta\theta$ can be directly used as the desired angle for the steered wheels. In fact, the steering command for ALX is the product of $\Delta\theta$ and a gain factor, which is empirically determined based on the maximum steering slew rate of ALX's front wheels.

The angular error $\Delta\theta$ (and thus the steering command) is calculated twenty times per second. This relatively fast upgrading and correction rate of the steering command ensures smooth heading control of ALX, especially when ALX operates at relatively low speeds (typically 0.3 - 0.5 m/s).

In the future, with better range sensors, we will try to increase the speed at which ALX (or other autonomous vehicles) operates (for example, over 30 MPH or 13.5 m/s). For operation at these speeds, the speed and the time interval between steering command corrections should also be taken into consideration for the inverse vehicle kinematics. A set of equation based on this situation has also been developed and verified in computer simulation. Details can be found in [34].

3.4 Overall Control Strategy

The discussion of the control strategy is keyed to the state transition diagram shown in Figure 3-12 below.

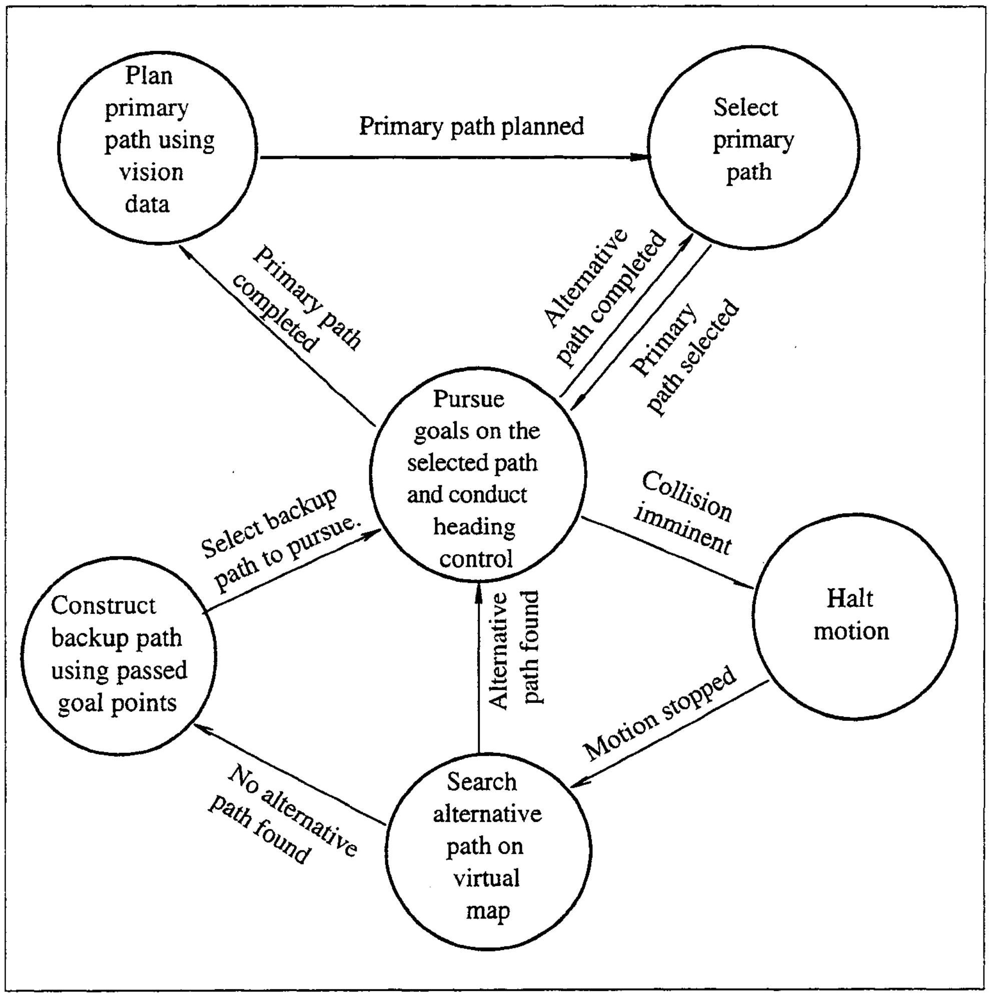

Plan primary path using vision data
Primary path planned
Select primary path
Primary path completed
Alternative path completed
Primary path selected
Pursue goals on the selected path and conduct heading control
Collision imminent
Halt motion
Select backup path to pursue.
Construct backup path using passed goal points
Alternative path found
Motion stopped
No alternative path found
Search alternative path on virtual map

Figure 3-12. State transition diagram of overall strategy.

The overall path tracking algorithm is based on the operation of goal stacks. There are two primary goal stacks, the *primary forward stack* and the *behind stack.* ALX attempts to travel towards the top goal on the primary forward stack. As each goal is attained, it is then pushed onto the behind stack. Other stacks come into play as described in detail below. Initially, both stacks are empty.

The overall algorithm is as follows.

A. Plan primary path.

From the received vision information, ALX generates a set of goals which define a primary forward path for the vehicle and puts these goals into the primary forward goal stack, with the nearest goal at the top of the stack.

B. Select primary path.

ALX selects the primary forward stack as the current goal stack.

C. Pursue goals on the selected path.

ALX pops a goal point from the current goal stack and attempts to drive towards it by using the pure-pursuit method described in Section 3-4.

If the current goal stack is empty and the current goal stack is the primary goal stack, go to step A.

If the current goal stack is empty and the current goal stack is not the primary forward stack, and ALX was executing an alternative path after detected obstacles ahead, go to step B.

If the current goal stack is empty and the current goal stack is not the primary forward stack, and ALX was backing up, go to step G.

As each goal point is attained while the robot is traveling forward, the attained goal is pushed onto the behind stack.

D. Update vision information (at all states).

ALX continuously runs a process in the background to receive vision data packages transmitted through the Ethernet from the vision processing unit. Coordinate transformation is conducted for the points on the reconstructed roadway edges. New primary forward goals are generated, and the primary forward goal stack is flushed. However when this occurs, the current goal is not changed to maintain smooth heading control.

E. Fire all ultrasonic range sensors (at all states).

As ALX travels, the ultrasonic range sensors are continually fired. A virtual map is updated as described in Section 4-3. Ultrasonic range returns are also used to detect imminent collisions.

F. Halt motion.

If a collision is deemed imminent, ALX stops. If the current goal stack is not the primary forward stack, the current goal stack is discarded.

Note that the only reason for the robot to stop completely are safety considerations: the ultrasonic range sensor has a very limited reliable detection range for outdoor use (less than 3 meters when experiencing a strong head-wind), and it takes considerable time (usually about 5 seconds) for the computer to compute an alternative path. Within such limited distance to the obstacles, we prefer that ALX stay still while it computes an alternative path.

G. Search for alternative path.

If an alternative path needs to be computed, a search is made in the virtual ultrasonic range map for an alternative path around the obstacle. Obstacle avoidance path searching scheme can be found in Section 4-2.

If an alternative path is found, it is stored in a secondary forward goal stack. This secondary forward goal stack is set to be the current goal stack, and operation is resumed at step C.

H. Construct backup goals.

If no alternative path is found, a secondary goal stack is constructed from the behind stack which stores the goals attained by ALX. A distance equal to the wheelbase is measured piecewise backwards along the path ALX just traveled. This is based on the assumption that it is still trafficable (i.e., there is no dynamically moving obstacles).

After backing is completed, another search for an alternative path is conducted, and the process described above is repeated. If ALX is forced to back up all the way to the starting point, the robot is deemed to be lost, and the algorithm halts.

I. Termination.

There is no normal terminating condition for this algorithm.

This strategy is relatively straightforward and computationally inexpensive for the computer resource (a MC68020 operating at 20M Hz) on ALX. It has been implemented on ALX and tested in various situations as reported in Section 5. It is our belief that if the processing unit is replaced with a faster computer, and the ultrasonic range sensors are replaced with sensors capable of longer detection range, smoother control behavior could be achieved. For example, ALX would be able to slow down and not have to stop completely while computing alternative paths after it detects obstacles ahead.

CHAPTER 4
OBSTACLE DETECTION AND COLLISION AVOIDANCE

One of the key issues for ALX's autonomous operation is to ensure its ability of detecting imminent obstacles ahead in the roadway and planning a detouring path accordingly. The on-board ultrasonic range sensors provide spatial information of ALX's local environment, and this information is registered into a *virtual map*. To correctly interpret this information, we developed a multi-value labeling scheme to remove some systematic errors caused by the relatively wide beam width of the ultrasonic sensors. Once ALX is determined to be obstructed, a graphic binary-tree searching approach is conducted on the virtual map to find an alternative path. In this section, we describe the virtual map concept, the multi-labeling scheme, and the alternative path searching method.

4.1 Spatial Information Sensing

An autonomous mobile vehicle like ALX should have the ability of constructing a database of its ambient spatial information on-the-fly during its autonomous operation, and the ability of finding alternative paths by using such a database when obstructed in order to reach its predefined destination. On ALX, the spatial information is collected by the on-board ultrasonic range sensors.

The spatial information processing algorithm is based on the concept of a *virtual map*, which is a representation of the local area around ALX (Figure 4-1). The virtual map is constructed by a two-dimensional (2-D) array, and each unit of this array is called a *cell*.. Because ALX is currently designed to operate only on a relatively flat horizontal surface, all the ultrasonic echoes are projected onto this 2-D Cartesian map, and information are registered into the cells. The virtual map is designed to be a square area about twice the length of the maximum sensing

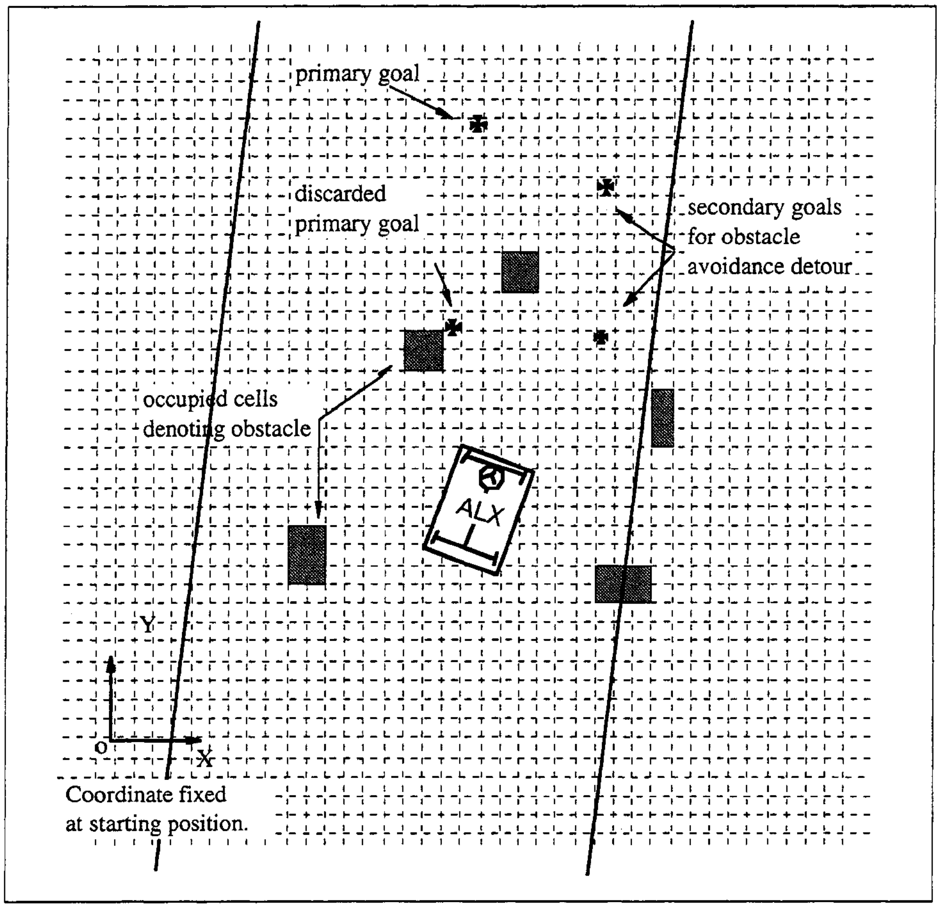


Figure 4-1. The Virtual Map (cells are not drawn to scale).

range along each edge. The virtual map "scrolls" under the robot as it moves. As ALX moves along, new "uncharted" areas appear on the leading edge of the map. Traversed areas "fall off" the trailing edge of the map, and all data associated with that part of the world is "forgotten." The rationale for this is the short time frame of accurate position data provided by the on board dead reckoning system. If ALX returns to an area previously visited, the position estimates will probably have accumulated enough error to make the saved data worthless.

The purpose of constructing the virtual map is to determine the local trafficability; therefore, the map should have accurately and clearly delineated open spaces and obstacles. One problem we encountered while using the ultrasonic range sensors is a variety of systematic errors which mainly result from incorrect and inconsistent interpretations of the raw ultrasonic range sensor data. This is largely due to the relatively wide beam width. To identify and remove this systematic errors arising, we have developed a deterministic method based on a multi-value labeling algorithm, which will be described later in this section.

4.1.1 Systematic Errors

The ultrasonic sensor returns a radial measure of the distance to the nearest object within the range of detection. The distance is found by measuring the time between the transmission of a pulse and the reception of its echo.

The ultrasonic range sensor can give rise to a variety of systematic errors ([19], [20], [35]). By systematic errors, we mean those errors that result from deterministic incorrect or biased interpretations of the raw data during processing. The systematic processing errors depend upon the beam width or resolution, the radiated power and sensing threshold, and the environmental geometry and surface properties of the objects being sensed. We do not have a priori information about the environment and the properties of obstacle surfaces. Therefore, we cannot model the physics of the sensing process sufficiently well to remove the errors prior to processing. Instead, we observe that there are several possible interpretations of the data from a given isolated scan. When combining data from different sensing positions or shooting angles, erroneous initial interpretations will give rise to conflicts. Whenever this happens, we can use physical argument to guide the replacement of erroneous interpretations with those that are consistent with the new information.

The principal systematic errors considered herein are those related to the relative broad beam width (about 15°) of the ultrasonic sensors. Figure 4-2 shows a schematic representations of some of the possible interpretations of a sensor return from a single scan angle. The arcs drawn in the figure denote the angular width of the ultrasonic sensor beam which was taken as 15°. These arcs are shown superimposed upon the virtual map.

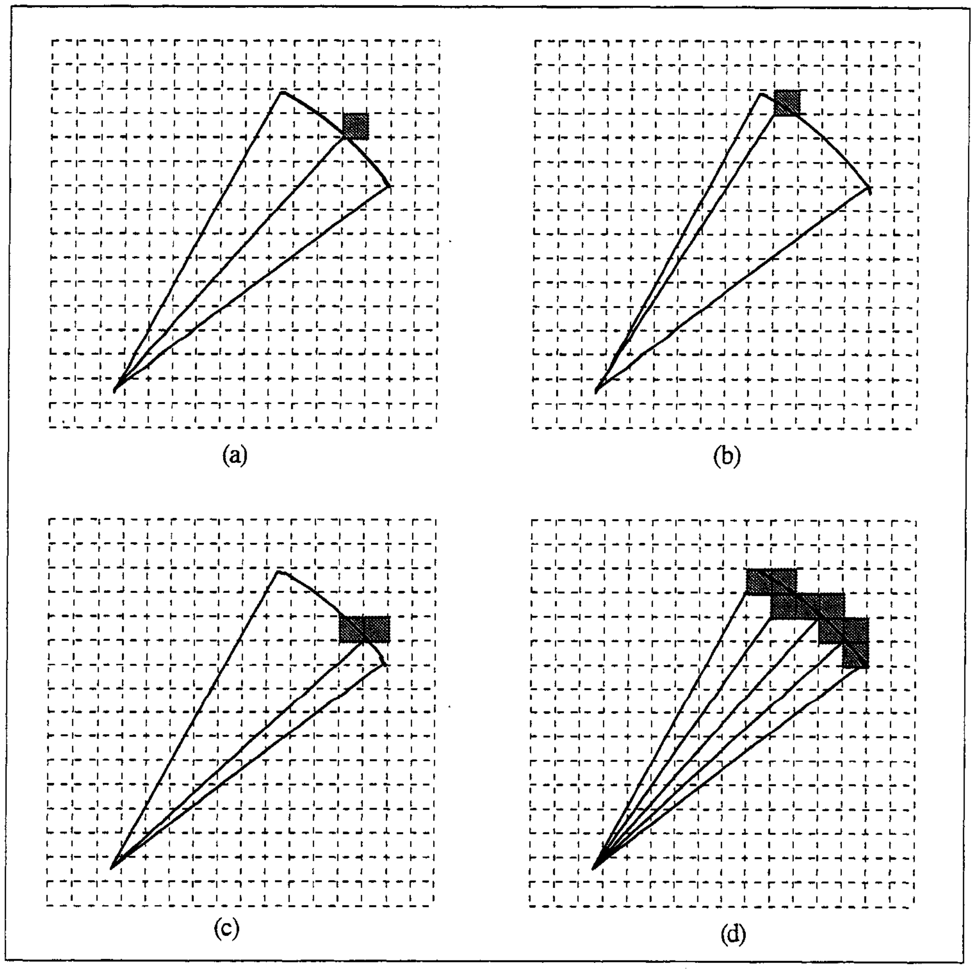


Figure 4-2. Four possible interpretations of an isolated return from a sensor scan. Arcs are not drawn to scale in this and the following figures.

In processing sparse data, we adopt the interpretation depicted in Figure 4-2 (d), in which each of the pixels intercepted by the arc is taken to be the one best representing the obstacle. Underlying the selection of this interpretation is a world view that ALX's environment consists of extended obstacles (relative to the geometric size of a cell which is 0.2 x 0.2 meters). This world view may be contrasted to one for a world containing only well-separated point objects. In such a point-object world, possibilities of interpretation depicted in Figure 4-2 (a) to 4-2 (c) would be more appropriate.

Under this interpretation, if a sensor "hits", i.e., the returned echo is less than the

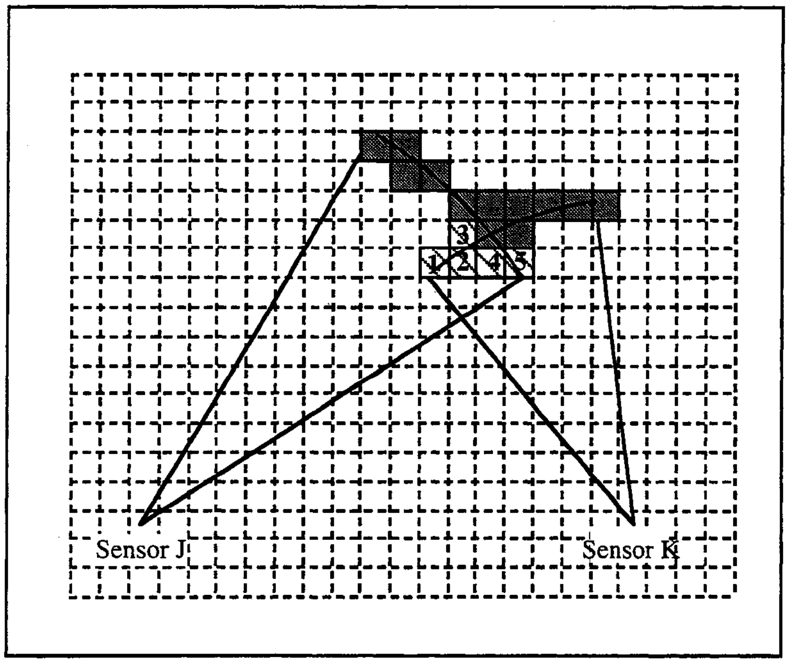


Figure 4-3. Conflict label assignments. Numbered cells receive conflict label assignments from different sensor scans.

maximum detectable range of the sensor, the area within the sensor footprint is divided into two regions: one is an empty region through which the sensor beam must pass, the other is an occupied region where the sensor beam stops. However, with data from different sensing positions and angles, interpretations for pixels in overlapped areas may give rise to conflicts. For example, in Figure 4-3 (a), ALX has two scans from two different observation points, overlapping each other to a certain extent. There is conflict in the common region of interest.

To resolve this conflict and achieve a consistent interpretation of the sensor data, instead of using stochastic schemes such as the grid certainty method [22], we were motivated by the observation that it would be useful to have simple, deterministic methods for the identification and removal of these systematic errors arising from the processing of sparse sensor data. In this regard, we have successfully adapted an algorithm which first effectively determines the cells in the virtual map covered by a single scan, and then use a multi-value assignment scheme to assign a label to each cell to resolve possible conflict between current assignment and assignment from previous sensor scans.

4.1.2 Sensor Information Mapping

In the following we present the method to determine which of the cells in the virtual map are affected by an isolated ultrasonic range sensor scan.

Calculating an ultrasonic range cone within the virtual map is a multi-step process. The overall goal is to select all the cells within the virtual map that are mostly contained within the ultrasonic range cone. In [19], Beckerman solved this problem by sweeping multiple "pencil beams" through the sensor cone area. This is computationally inefficient if complete coverage is desired, since some of the cells are scanned more than once. If it is important to mark only completely-contained cones, the sweeping technique does not lend itself to the determination that a map cell *is* or *is not* completely contained.

To improve computational efficiency, we developed a cell-selecting algorithm which guarantees that each map cell is visited only *once*, and that *all* cells contained in the sensed cone area are visited.

This cell-selecting algorithm is summarized here first, and presented in detail below. Step one is to determine what "sector" the beam falls into. This helps to focus algorithms for finding vertical and horizontal extents. Step two is to determine the extent of the beam along the Y axis (vertical extent). Step three is to iterate through all rows contained in the virtual map, and, for each row, determined the extent in the X (horizontal) direction.

Step 1. Determining Beam Sector.

Beam sector is determined exclusively from the angle of the beam centerline and the width of the beam. Refer to Figure 4-4 for sectors and sector number assignments.

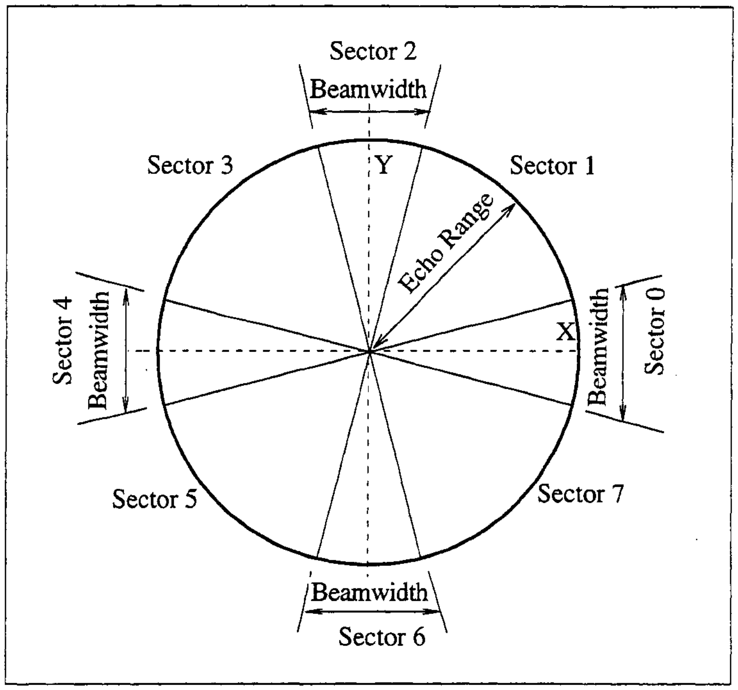


Figure 4-4. Assignments of sector numbers.

The numeric values of the sector assignments are unimportant except for use as reference in following calculation steps.

Step 2. Determining Vertical Extent

Vertical extent is determined differently for each sector. Values needed to be given to calculate vertical extent are:

y_c	-y ordinate of beam origin in virtual map
l	-range of beam (length of ultrasonic range sensing cone) in virtual map length unit
q	-angle of beam centerline
s	-sector number
b	-half beam width

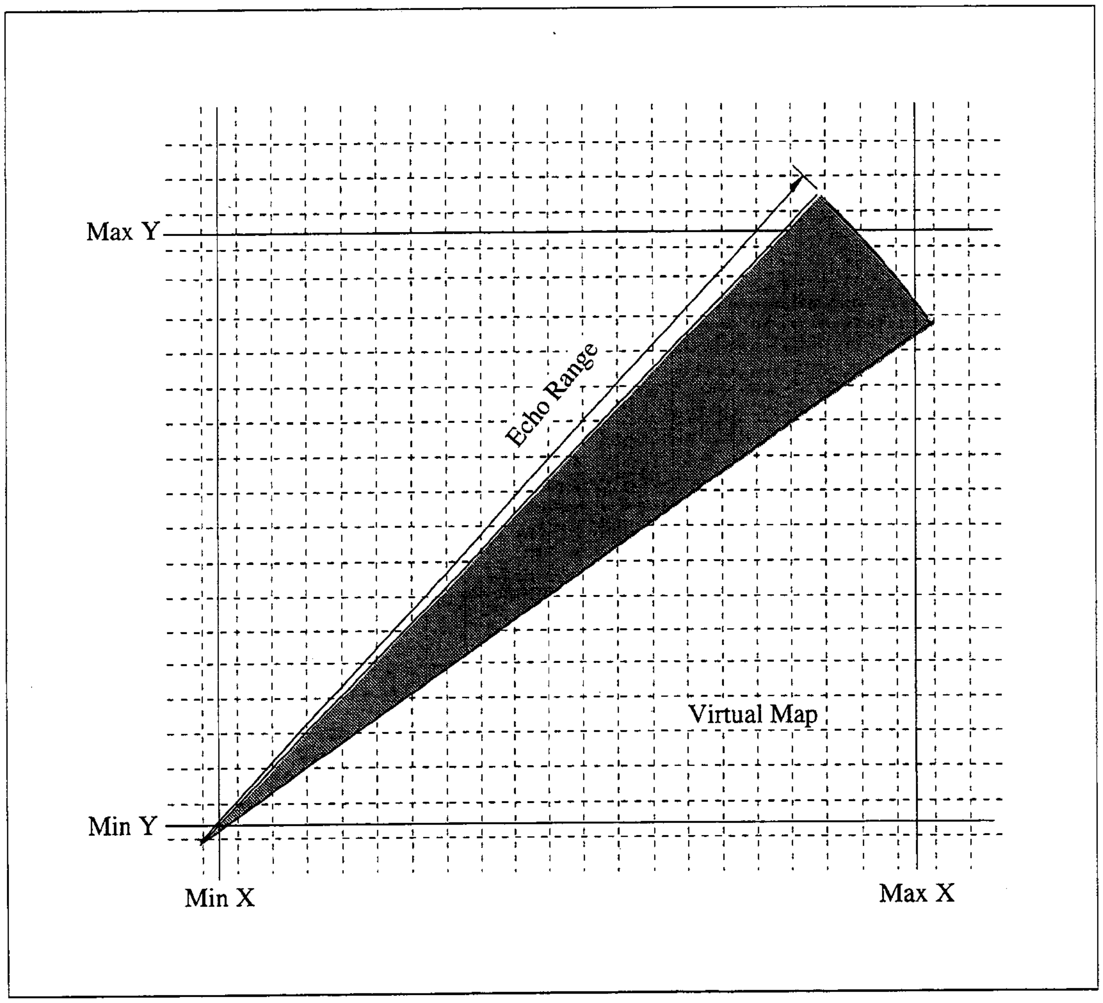


Figure 4-5. Overall bounding rectangle of a sensor beam in sector 1.

First, one calculates the y ordinate (Figure 4-5) of the endpoints of the right and left beam edges. The right beam edge is the edge clockwise from the beam centerline. The left beam edge is the edge counterclockwise from the beam centerline. The y ordinates of the intersections of the circle (Figure 4-6) with left and right beam edges are denoted as y_l and y_r, respectively. The "trunc" function shown below means "truncate towards zero." Next, y_{min} and y_{max} are calculated using a sector-dependent algorithm as follows:

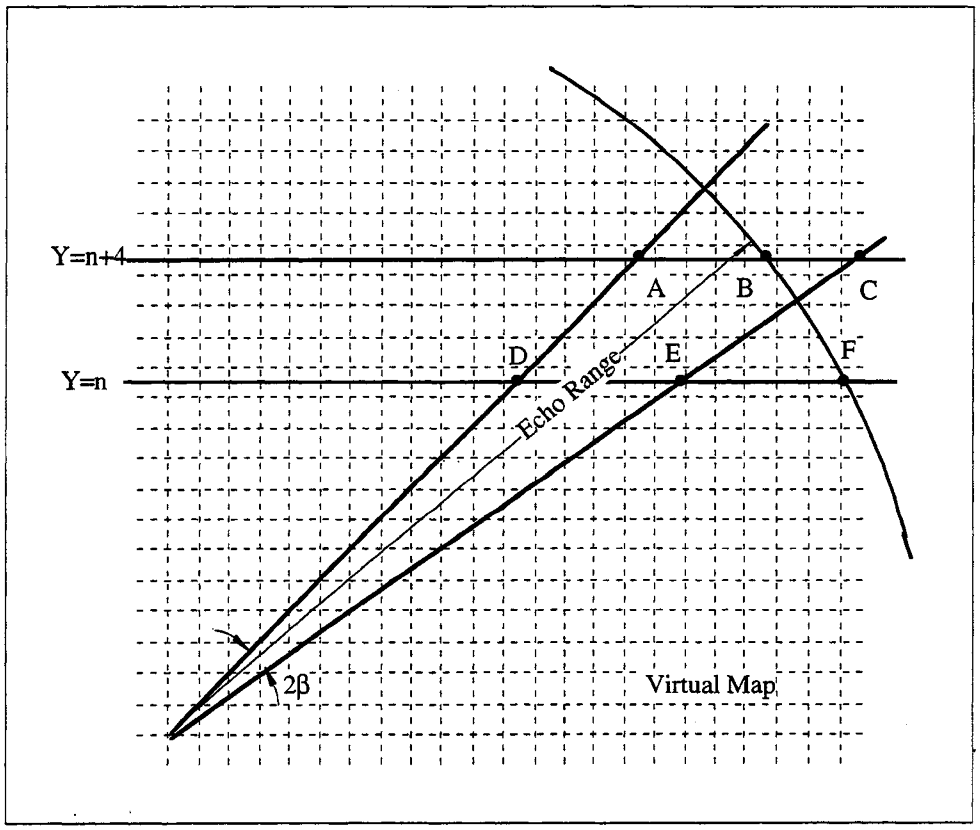


Figure 4-6. Calculation of horizontal extent.

$s = 0$:

$y_r = y_c$ "+" $trunc\,(\,l \bullet sin\,(q - b\,)\,)$ *(4-1)*

$y_l = y_c + trunc\,(\,l \bullet sin\,(q - b\,)\,)$ (4-2)

$y_{max} = y_l$ (4-3)

$y_{min} = y_r$ (4-4)

$s = 1$:

$$ymax = yl \quad (4\text{-}5)$$

$$ymin = yc \quad (4\text{-}6)$$

$s = 2$:

$$ymax = yc + trunc(\,l\,) \quad (4\text{-}7)$$

$$ymin = yc \quad (4\text{-}8)$$

$s = 3$:

$$ymax = yr \quad (4\text{-}9)$$

$$ymin = yc \quad (4\text{-}10)$$

$s = 4$:

$$ymax = yr \quad (4\text{-}11)$$

$$ymin = yl \quad (4\text{-}12)$$

$s = 5$:

$$ymax = yc \quad (4\text{-}13)$$

$$ymin = yl \quad (4\text{-}14)$$

$s = 6$:

$$ymax = yc \quad (4\text{-}15)$$

$$ymin = yc - trunc(\,l\,) \quad (4\text{-}16)$$

$s = 7$:

$$ymax = yc \quad (4\text{-}17)$$

$$ymin = yr \quad (4\text{-}18)$$

Step 3. Determining Horizontal Extent

Calculating the horizontal (x) extent of a cone for a given value of y depends on the sector containing the cone. Refer to Figure 4-3 for clarifying diagrams. Values needed to be given to calculate horizontal extent are:

(x_c, y_c)	-y ordinate of beam origin in virtual map
l	-range of beam in virtual map length unit
q	-angle of beam centerline
s	-sector number
b	-half beamwidth
y	- y (row) number in virtual map

Four calculations determine the possible values for x_{min} and x_{max}:

Intersection of y with the left beam edge x_r

$$x_l = x_c + trunc(\frac{(y-y_c)}{tan(\theta+\beta)}) \qquad (4\text{-}19)$$

Intersection of y with the right beam edge x_r:

$$x_r = x_c + trunc(\frac{(y-y_c)}{tan(\theta-\beta)}) \qquad 4\text{-}20)$$

Intersection of y with the circle, x_{cl} and x_{cr} (left and right):

$$x_{cl} = x_c + \sqrt{\lambda^2 + (y-y_c)^2}) \qquad (4\text{-}21)$$

$$x_{cr} = x_c - \sqrt{\lambda^2 + (y-y_c)^2}) \qquad (4\text{-}22)$$

The algorithm then finds x_{min} and x_{max} which varies by sector s:

$s = 0$:

$$x_{min} = max\ (x_l, x_r) \qquad (4\text{-}23)$$

$$x_{max} = x_{cr} \qquad (4\text{-}24)$$

$s = 1$:

$$x_{min} = x_l \qquad (4\text{-}25)$$

$$x_{max} = min\ (x_r, x_{cr}) \qquad (4\text{-}26)$$

$s = 2$:

$$x_{min} = max\ (x_{cl}, x_l) \qquad (4\text{-}27)$$

$$x_{max} = min\ (x_r, x_{cr}) \qquad (4\text{-}28)$$

$s = 3$:

$$x_{min} = max\,(x_{cl}, x_l) \quad (4\text{-}29)$$

$$x_{max} = x_r \quad (4\text{-}30)$$

$s = 4$:

$$x_{min} = x_{cl} \quad (4\text{-}31)$$

$$x_{max} = min\,(x_l, x_r) \quad (4\text{-}32)$$

$s = 5$:

$$x_{min} = max\,(x_r, x_{cr}) \quad (4\text{-}33)$$

$$x_{max} = x_l \quad (4\text{-}34)$$

$s = 6$:

$$x_{min} = max\,(x_r, x_{cr}) \quad (4\text{-}35)$$

$$x_{max} = min\,(x_l, x_{cl}) \quad (4\text{-}36)$$

$s = 7$:

$$x_{min} = x_r \quad (4\text{-}37)$$

$$x_{max} = min\,(x_l, x_{cr}) \quad (4\text{-}38)$$

4.1.3 Multi-Value Labeling Scheme

To resolve conflict in interpretation of the ultrasonic range sensor information, various stochastic methods have been used ([21], [22], [23]). In these methods, at each sensor scan, the profile of probability of the cells in the sensor cone has to be recalculated, and then every cell has to be updated. This is computationally expensive. In contrast, the multi-value labeling scheme described below is much less computationally intensive, yet it is able to resolve conflict between current label assignment and assignment from previous sensor scans. This gives sufficient treatment to systematic errors primarily caused by the ultrasonic range sensor's wide beamwidth.

At each scan, given the boundaries of the sensor cone, the virtual map is raster-scanned by using the method described in 4.1.2. Each cell inside this cone on the virtual map can be given a preliminary label as follows: if the sensor returns a "hit" and the distance from the origin of the cone to the center of the cell under consideration is greater than the sensed range, then the cell is given an assignment of "OCCUPIED". Otherwise, the cell is assigned to be "EMPTY." For example, in Figure 4-3, for Sensor K, both cell 1 and cell 2 are labeled as "OCCUPIED," while both cell 4 and cell 5 (well inside the boundary) are labeled as "EMPTY".

In addition to a label, each cell is also assigned an attribute, which is the range of the sensor's actual return. This attribute is used in a relabeling algorithm to determine which sensor was closer to the detected obstacle. The closer the range sensor is to the object, the more confidence we have in the returned range value (provided that the returned range is greater than the sensor's minimum detectable range). Thus, when conflicting assignments arise, the cell label is assigned based on the sensor which is closest. For example, in Figure 4-3, the cells receiving conflicting label assignments will be finally labeled in favor of Sensor K.

The current assignments are then resolved with the virtual map as follows:

A. If the cell is previously labeled "UNKNOWN", then it is re-labeled with the current assignment, and the attribute of the cell is set to the range of the current sensor range return value.

B. If the cell is previously labeled "EMPTY", and the current assignment is "EMPTY", then the attribute of the cell is updated to the current sensor range return value.

C. If the cell is previous labeled "OCCUPIED", and the current assignment is "OCCUPIED", then the attribute of the cell is updated to the current sensor rage return value.

D. If the cell is previous labeled "OCCUPIED", and the current assignment is "EMPTY" (for example, cell 4 and cell 5 in Sensor K's scan shown in Figure 4-3), then if the current sensor range return value is less than the cell's previous attribute (which is the echo return from Scan J in Figure 4-3, for example) times a factor A_{EMPTY} (this means a “closer look”), then the cell is re-labeled “EMPTY”, and the attribute of the cell is updated to the current sensor range return value; otherwise the cell’s attribute and label assignment remain.

E. If the cell is previously labeled "EMPTY", and the current assignment is "OCCUPIED" (for example, cell 1, cell 2 and cell 3 in Sensor K's scan shown in Figure 4-3), then if the current sensor range return value is less than the cell’s previous attribute times a factor $A_{OCCUPIED}$, then the cell is re-labeled “OCCUPIED”, and the attribute of the cell is updated to the current sensor range return value.

The re-labeling algorithm is based on our observation that closer the ultrasonic range sensor to the object, the more confidence we have in the returned range value. The factor $A_{OCCUPIED}$ is chosen to be less than the factor A_{EMPTY}, to favor marking obstacles. In our experiments, we have determined empirically that the following assignment gives good results: $A_{OCCUPIED}=0.8$ and $A_{EMPTY}=0.9$.

The multi-labeling scheme is much less computationally intensive compared to the stochastic methods, yet adequately handles systematic errors primarily caused by the range sensor's relatively wide beamwidth. For example, the conflicts depicted in Figure 4-3 are resolved after the re-labeling process, as shown in Figure 4-7.

Note that not all systematic errors can be treated in this way, and such a methodology is best suited to the case where the data is sparse, the patterns of conflict are simple, and the corrections are physically unambiguous. In order to overcome these limitations, we have successfully developed a system that can generate topologically correct maps on line in real time for ultrasonic range sensors. The methodology, based on self-organizing neural networks, is described by Morrellas et al. [37].

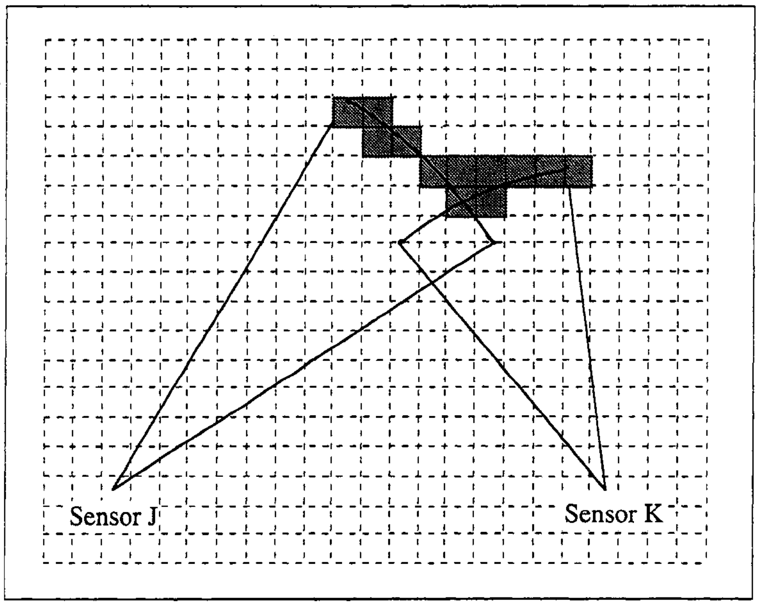


Figure 4-7. Consistent labeling. Conflict labels shown in Figure 4.3 are resolved during the multi-value labeling process.

4.2 Alternative Path Searching

There are many proposed obstacle avoidance methods. The *edge-detection method* ([3], [38], [39], [40]) is one popular method from early research. In this method, an algorithm tries to determine the position of the vertical edges of obstacle and steers the robot around either one of the "visible" edges. The line connecting two visible edges is considered to represent one of the boundaries of the obstacles. A disadvantage of this method is its sensitivity to sensor accuracy/inaccuracy, and ultrasonic sensors present shortcomings in this respect (as discussed earlier in this section). Errors in ultrasonic range sensor readings can cause this algorithm to determine the existence of an edge at a completely erroneous location, often resulting in highly unlikely paths.

Another popular method used by researchers for obstacle avoidance is the *potential fields method.* The idea of imaginary forces acting on a robot has been suggested by Khatib [41]. In this method, obstacles exert repulsive forces, while the target imposes an attractive force to the robot. A resultant force vector R, comprising of the sum of a target-directed attractive force and repulsive forces from obstacles, is calculated for a given robot position. With R as the accelerating force acting on the robot, the robot's new position for a given time interval is calculated, and the algorithm is repeated. Various approaches based on this force field concept have been proposed (for example, see [42], [43], and [44]). These methods usually require extensive computation power because of the representation conversion process or calculation of force fields involved. Considering that the navigation of the mobile robot should be performed in real-time this could be an important shortcoming (especially when the major computer resource on board ALX is a 25 MHz 68020 microprocessor).

Rather than try to determine the obstacle's exact visible edges (which is used in the edge-detecting method), or use computationally expensive algorithms (as the various methods using potential field concept), we have developed a graphic binary-tree search algorithm for ALX's alternative path planning. This method utilizes the virtual map, which stores the occupancy of ALX's local environment and an "attracting" point which lies on the primary path of goal points. ALX tries to reach the attracting point while avoiding obstacles by searching for regions of "empty" cells. The algorithm tests each consecutive goal point by comparing the "footprint" of ALX with the occupancy of the virtual map under the footprint. If occupied, the algorithm tries a new goal point to the right and then to the left. If these are occupied, ALX backs up by one meter and iterates to find a new path around the occupied region. The algorithm has the following steps:

A A new goal point (the attracting point) is chosen along the primary forward goal stack's path. The attracting point is chosen to be a distance d measured piece wise along the primary forward goal path. Closer goal points are popped and discarded (Figure 4-8). The value of d can be selected based on environmental expectations (for example, the depth of the obstacle), or adjusted dynamically as the robot learns its environment. In our experiment, we used a fixed value that is about twice of the sensor maximum range.

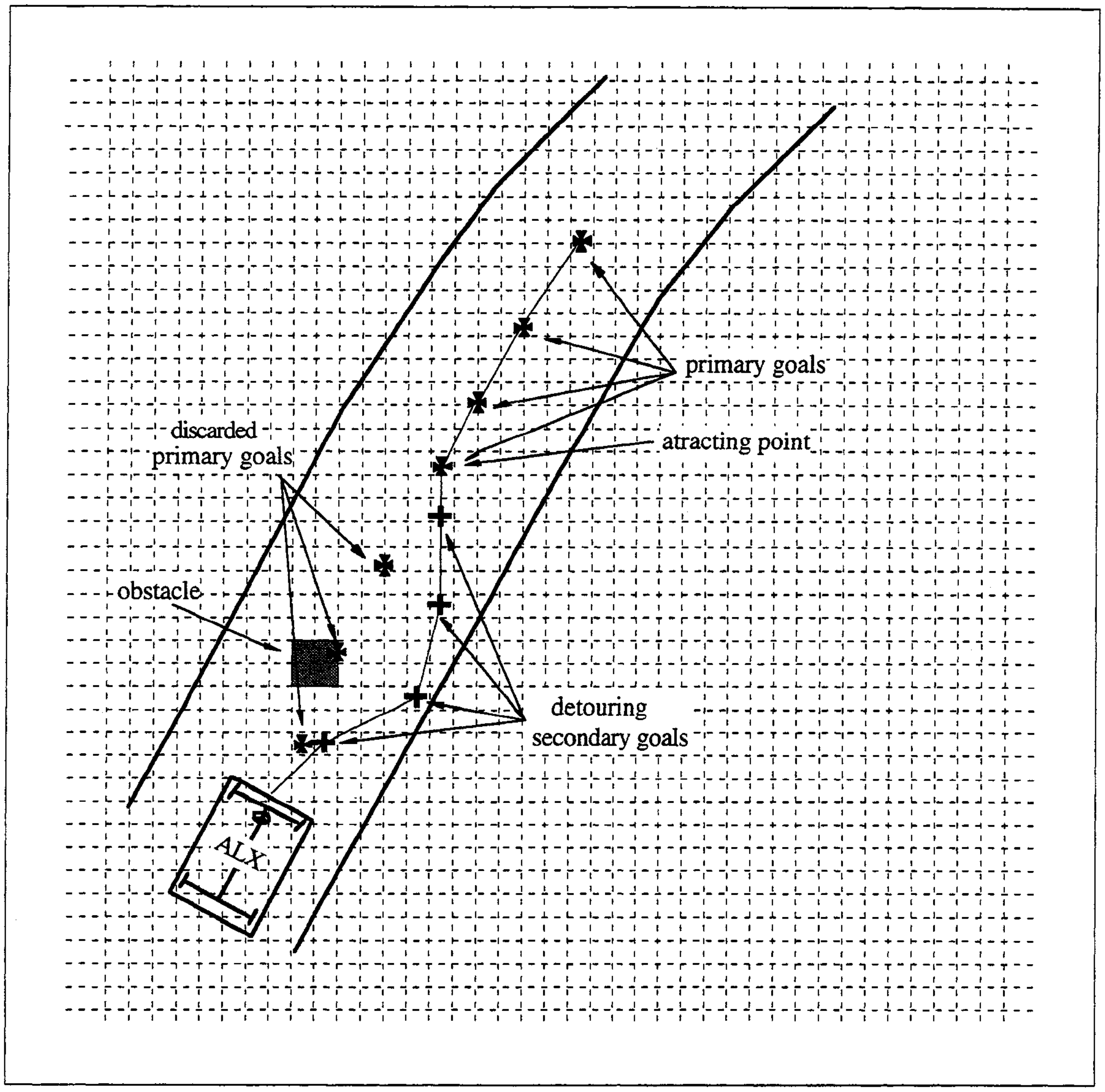


Figure 4-8. Detouring goals are generated during collision avoidance path planning.

B. ALX then uses a recursive *binary-tree* search algorithm (Figure 4-9) to find a new path. The robot's position and orientation are passed to a search function. A stack to save the found path, if any, is also available to this function. Each invocation of the search function does the following:

i. Recursively call with two new positions. The first position is calculated by moving a small distance and turning the robot towards the attracting point selected in step A. If in that direction ALX fails to find a path, the second position is calculated by turning the robot away from the attracting point selected in step **A**. In both cases, ALX is turned at a maximum turning radius.

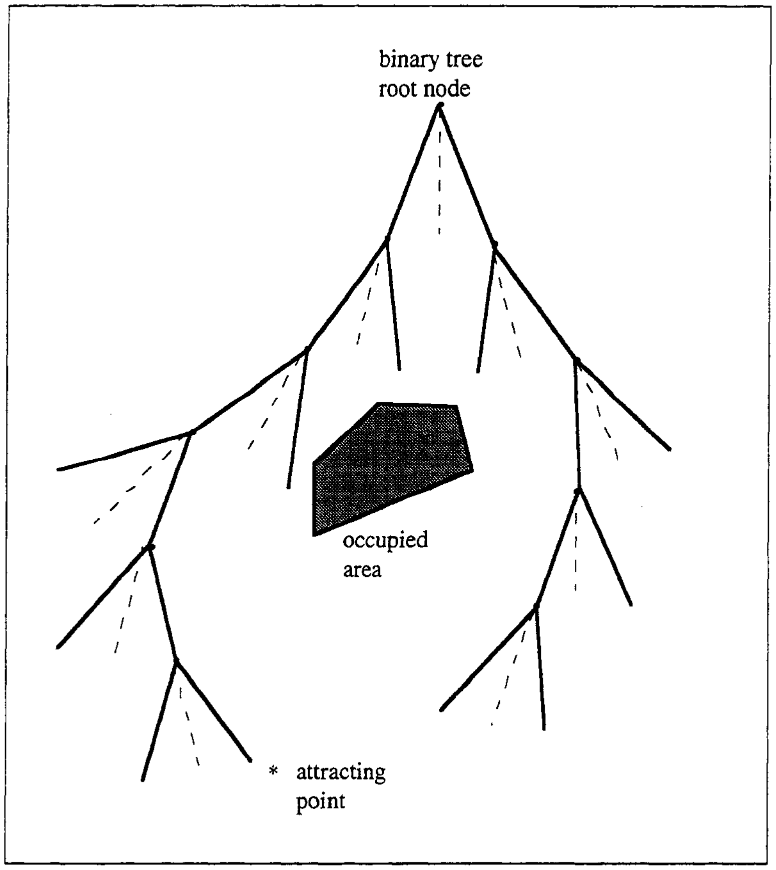


Figure 4-9. Binary-tree search method. At each node, the search can either turn left or turn right (binary decision). It uses the attracting point to determine which direction to try first. If both directions are blocked, it then has to go back to the parent node. (The turn-angle and search-step size are fixed for all nodes.)

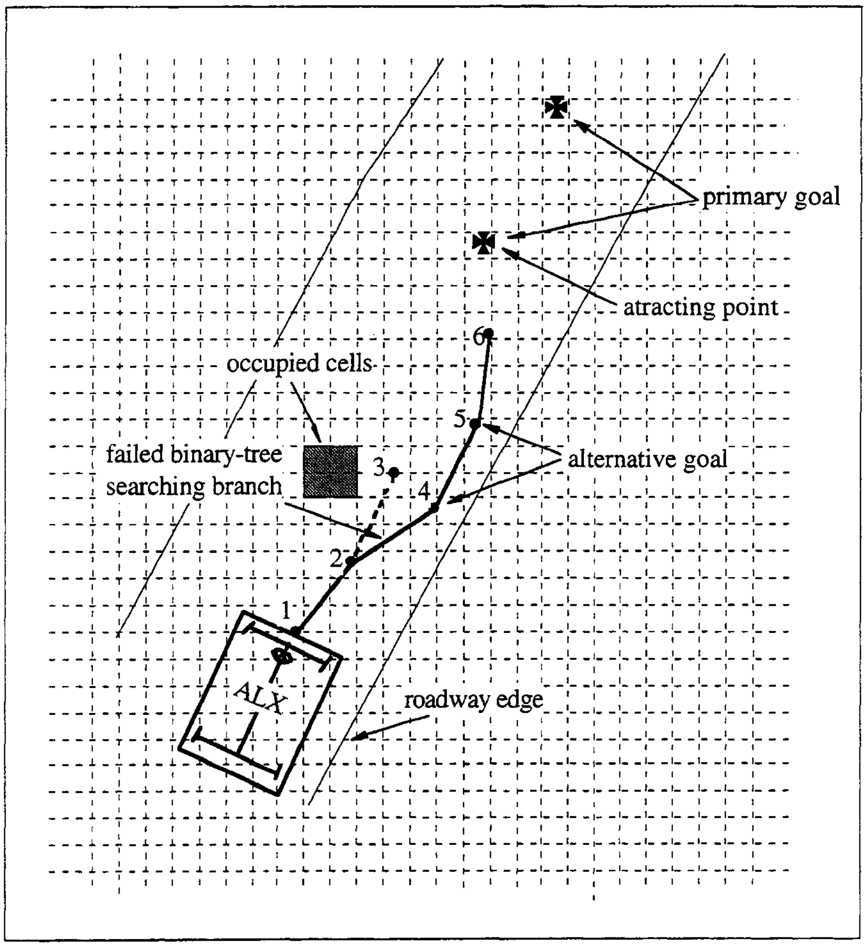


Figure 4-10. Apply the binary-tree search method on the virtual map.

ii. To judge a position to be potential goal point, a template of the "footprint" of ALX[1] is placed onto the virtual map (Figure 4-10), and the cells covered by the template are examined. If *any* cells are tested to be OCCUPIED, then this position is not a potential secondary goal point, and it cannot be used to construct an alternative path. The previous goal point which lead to this position during the search is then popped out from the temporary goal point stack and discarded. If *all* cells under the template are either EMPTY or UNKNOWN, then this position is treated as a potential secondary goal point, and is stored in a temporary goal point stack.

[1] The template must be slightly larger than the robot, because the robot sweeps an area larger than itself while turning.

iii. If the current position is determined to be a potential secondary goal point, and it is a piecewise distance of at least *d* from the point where the path-searching started, this position is pushed onto the temporary goal point stack, and the recursive path searching process is exited and a "successful" signal is returned.

iv. If the process fails to find a path in both binary directions extended from the point where the whole searching process starts, then a "failure" signal will be returned.

C. If the main process detects a "successful" signal from the recursive binary-tree alternative path searching process, it will use the temporary secondary goal point stack which is also returned, and construct an alternative path based on the points in this stack. On the other hand, if the returned signal is "failure", then ALX will back up a certain distance. To do this, it uses the goals saved in the passed-goal stack, and constructs a backing-up set of goal points. This operation is based on the assumption that the space it traversed is still trafficable; in other words, there are no dynamically moving obstacles.

D. If ALX is forced to back up all the way to its starting position, it signals that it is lost, and the autonomous operation is stopped.

This alternative path planning algorithm has several limitations:

1. It is not *complete*. There may be some usable paths that are not found because of the discrete nature of the algorithm.
2. Computation time is sensitive to the size of the step distance selected. A smaller step leads to exponentially-increasing computation time. A larger step misses more possible solutions.
3. The present implementation does not use the robot inverse kinematics equations to predict movement because the inverse kinematics require a look-ahead capability on the path. This would require a very sophisticated tree search algorithm to keep track of alternatives for at least two points in the search tree.

With all limitations, this algorithm still can be tuned to give usable results in real time. This alternative path planning algorithm was implemented and tested on ALX, which executed both collision avoidance and vision based lateral control algorithm simultaneously on a test track. A number of scenarios were tested, ranging from a single obstacle near the lane boundary (requiring only slight trajectory corrections) to multiple obstacles which completely blocked the marked lane (requiring a reversal of direction to maneuver around the obstacles). Experimental results are presented in Section 5.

CHAPTER 5
SIMULATION RESULTS

A computer simulation environment, *Navsim,* was first developed before the control programs were implemented on ALX. The simulation has realistic models of the robot's ultrasonic range sensor and camera and is programmed in the same manner as ALX. Control algorithms are first tested in the simulation before they are implemented on ALX. The simulation program has proven to be a valuable debugging tool.

The first version of the simulation was developed by Don Krantz [28]. This system was capable of detecting and avoiding obstacles while guiding the robotic vehicle through a set of predefined goals. Curt Olson [45] then adapted the simulation to a roadway environment, and the simulated vehicle was able to drive itself along a road track while detecting and avoiding obstacles in its path (Figure 5-1).

The following are some of the major features of the simulation program.

- A model of the ultrasonic range sensor. This model has the same configuration as the ultrasonic range sensors on ALX, including mounting position, orientation and maximum/minimum detecting range of each individual sensor. It calculates the distances between the vehicle and all the obstacles, and alerts the vehicle when any of these ranges are less than the maximum distance which a sensor can detect.
- A simulated dead-reckoning system. This system provides the world (x,y) coordinates of the vehicle and the orientation of the vehicle with respect to the starting frame.
- A simulated vision system. The simulated vision system provides road information which is the position and orientation of visible road edges. This information can then be passed to the vehicle so that it can make decisions on how to move.

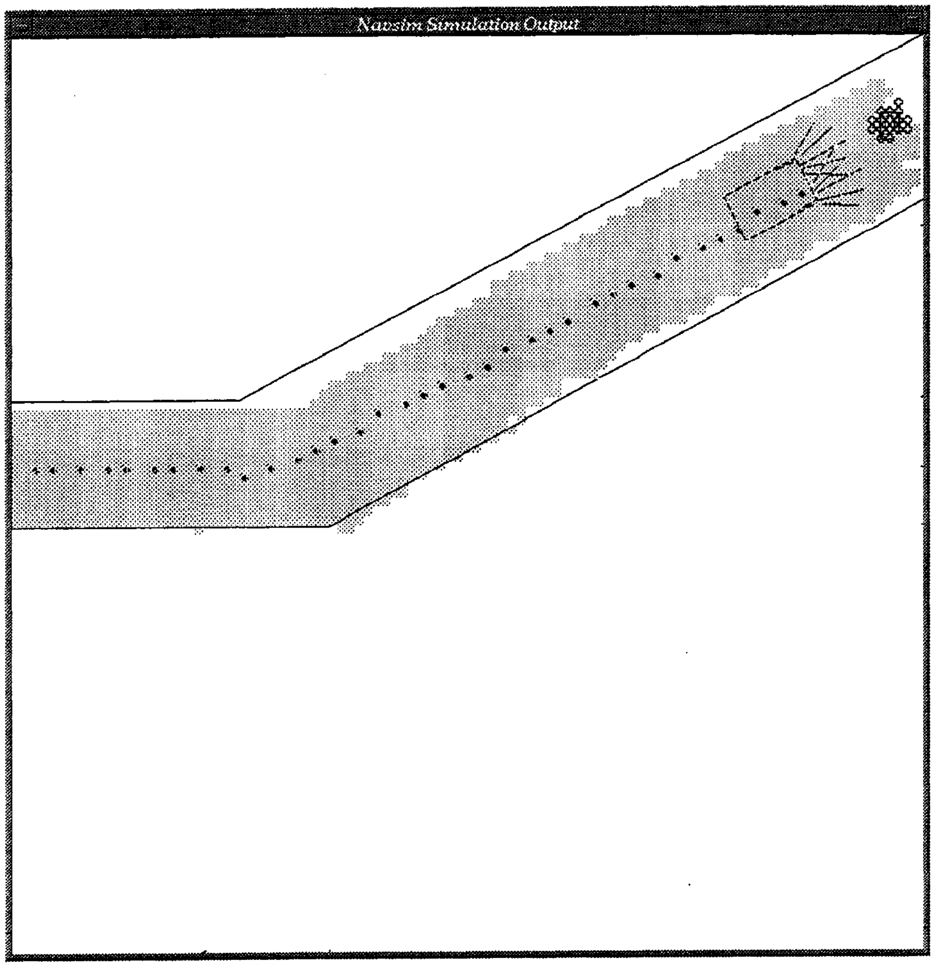
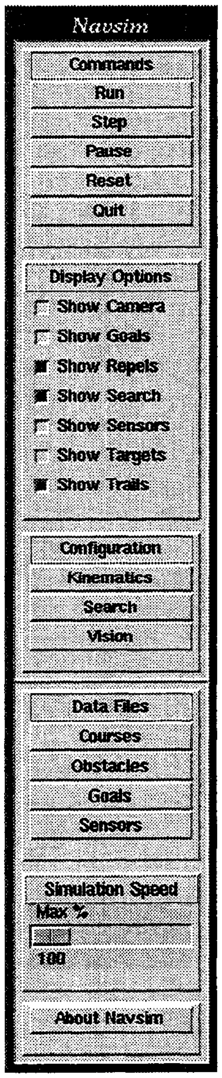


Figure 5-1. Output window of Navsim - a simulation program for ALX.

ALX is approaching an obstacle.

• User Interface. A front end graphic user interface (GUI) was developed by using a scripting language *Tool Command Language/Tool Kit.* It provides a Motif look and feel and includes buttons, menus, check boxes, scale widgets, etc. Figure 5-2 shows the appearance of the user interface.



- [ ] Show Camera
- [ ] Show Goals
- [x] Show Repels
- [x] Show Search
- [ ] Show Sensors
- [ ] Show Targets
- [x] Show Trails

Max %
100

Figure 5-2. Command window of Navsim. User can change simulation system configuration parameters by using this command window.

• Simulated Roadways. Roadways with various lengths and curvatures can be used in the simulation. In the simulation, a roadway is defined by its left and right edges. An edge can be approximated by a series of line segments, which are further defined by a series of points. These series of points can be saved in a file. In this way multiple courses can be defined, and the desired one can be selected at run time. While the output windows (Figure 5-1) provides detailed ALX movement information, a small overview window (Figure 5-3) displays current position of ALX on the simulated track.

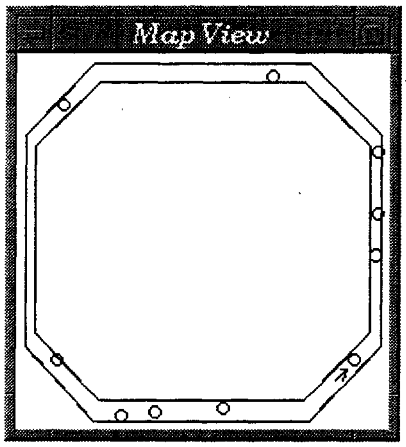


Figure 5-3. A small window displays the overview of the simulated roadway. The arrow denotes the current position of ALX.

A number of obstacles are placed on various positions on each track. Each obstacle is defined to occupy an area of 1 meter by 1 meter. Their positions are denoted by squares in the Navsim output window, and by circles in the Map View window.

Two examples of the simulation result are presented here.

In the first example, starting from Figure 5-1, ALX is approaching an obstacle placed almost in the middle of the roadway. The ultrasonic range sensors are shooting into the direction denoted by the straight lines starting from the front end of ALX. The gray areas are detected unoccupied areas, and the white areas are unknown areas. The small crosses represent the occupied area (i.e., occupied cells in the virtual map). Note that the ultrasonic range sensors have relatively large beamwidth (15 degrees), so a precise profile of the obstacle cannot be obtained in a finite number of ultrasonic range sensor scans, so the area covered by the small crosses is usually slightly larger than the area actually occupied by the obstacle. The small black dots denotes the trail of the center point of ALX's front end. Figure 5-4 and Figure 5-5 show ALX stopped before the obstacle, backed up a short distance for more room to maneuver, and then successfully determining an alternative path around the obstacle. After executing this alternative path, ALX continued its path tracking operation along the roadway.

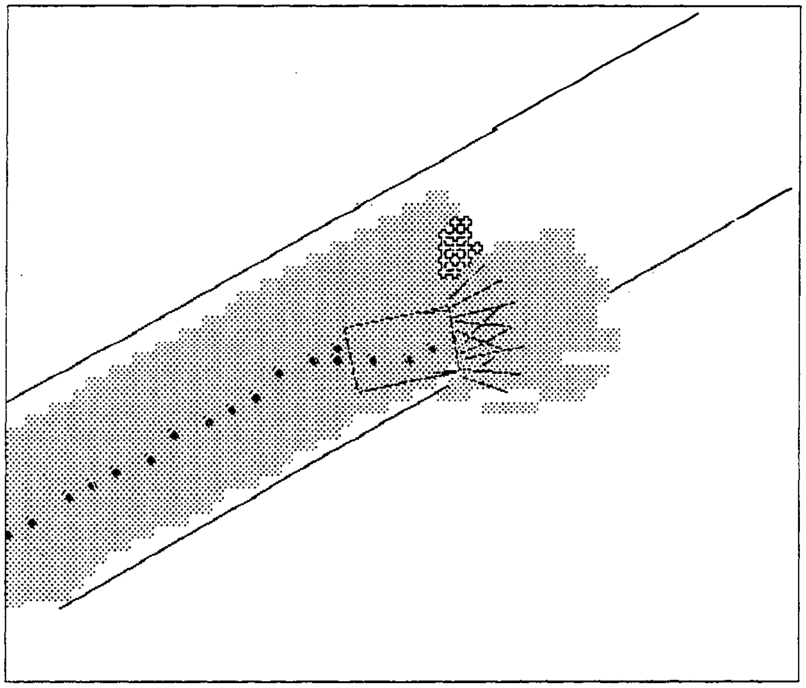


Figure 5-4. In simulation, ALX detected an obstacle and calculated an alternative path to go around the obstacle.

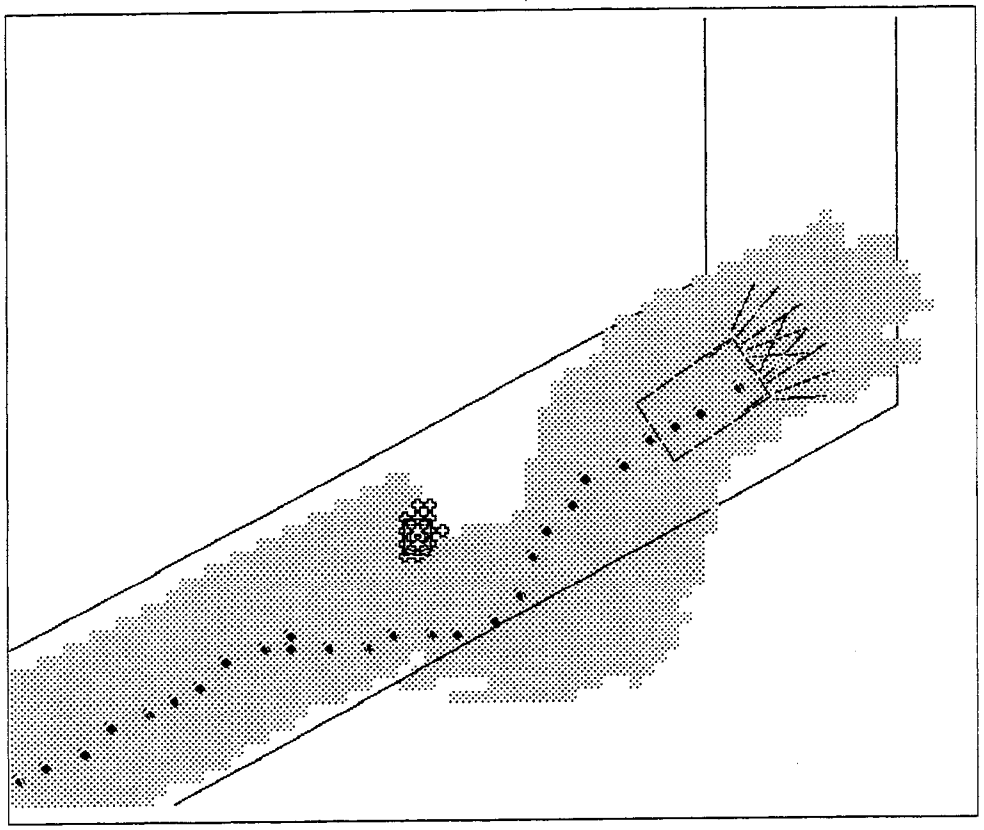


Figure 5-5. In simulation, ALX successfully avoided an obstacle and resumed path tracking.

In the second simulation example, as shown in Figure 5-6, ALX passed a roadway section which contained a number of obstacles. Because of the limited detection range of the ultrasonic range sensors, ALX was only able to "see" the immediate obstacle ahead, not all obstacles in the field. Therefore, no optimized path was generated for this obstacle field, and ALX continued to plan its path during the navigation process. Notice that if the obstacle is in the center of the road (for example, the first and third obstacles), then ALX might have to leave the road temporarily for a safe avoidance trajectory, and come back to the track after the obstacle avoidance operation is finished.

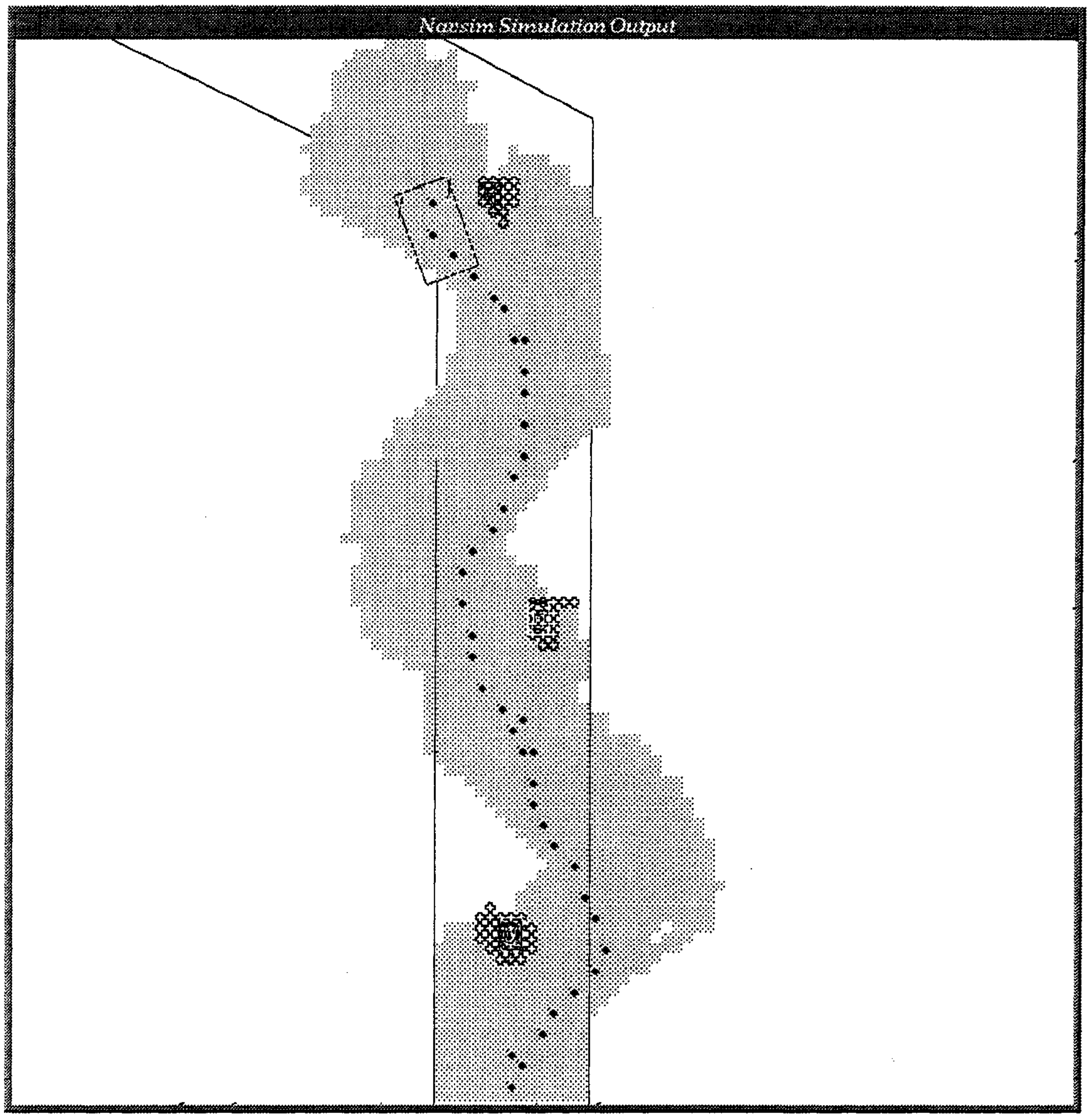


Figure 5-6. In simulation, ALX passed a roadway section which contained a number of obstacles.

As one can see, this simulation program works reasonably well. However, it cannot provide complete fidelity. Some of the limitations of the Navsim simulation program are listed as follows:

- On ALX, all control programs are executed under the VxWorks real time operating system. VxWorks allow these programs to break into tasks which have different priorities. VxWorks provides a preemptive scheduling mechanism which executes these tasks according to their priorities, and makes it appear that these tasks are running in parallel. In the simulation, all programs are executed under Unix and there is no multi-tasking mechanism used. All programs are executed sequentially. The simulation program cannot simulate the dynamic computing interaction between all the control programs, which is an important aspect of the ALX control architecture.

- In simulation, a pseudo-vision function is employed. This function uses the predefined track information to generate goals, i.e., it always provides accurate vision information. On the other hand, the real image processing unit on the ALX may not always be able to provide accurate roadway information, because ideal situations (e.g. good lighting conditions, clean road surfaces, etc.) cannot be guaranteed.
- In the simulation program, a set of kinematic equations are used to update the vehicle's position and orientation in each calculation cycle. However, these equations cannot accurately model the real golf cart, because many factors (for example, the dynamics of the golf cart) are not taken into consideration, and many assumptions are used to simplify the model. As a result, although the same control equations are used, the simulation and the real vehicle might generate similar, but not identical trajectories.
- In the simulation, accurate numerical calculations are used to simulate the physical sensor signals (for example, the outputs of the fluxgate magnetometer and the returns of the ultrasonic range sensors). However, on ALX, these signals may be affected by a number of factors.. Possible disturbance factors can include a slightly fluctuating local magnetic field for the magnetometer, and a gusting-wind for the ultrasonic range sensors. Since ALX responds to its environment based on the information provided by the sensors, even under the exactly the same testing scenario, slight difference will exist in ALX's behaviors between the simulation and the real test results.

With these limitations in mind, this simulation program still has proven to be a valuable debugging tools for initial testing of the ALX control algorithms and various system configurations. It is especially useful during the long winter in Minnesota when outdoor experiments are not possible.

CHAPTER 6

EXPERIMENTAL RESULTS

In this section, we present some of outdoor test results of ALX's autonomous operation.

On our test site, we conducted three phases of experiments to test ALX's control algorithms. The first phase is roadway following without any obstacles in the roadway. This phase is designed mainly to test the vision function and goal generation algorithm. The second phase is obstacle avoidance while moving towards predefined goal points. In this phase, no vision processing is involved and the major issue here is to test the obstacle detection and collision avoidance algorithms. The third part is a combination of the first two: ALX follows roadways which contain obstacles either on the side or near the middle of the roadway.

6.1 Roadway Following Test

Figure 6-1 and 6-2 illustrate ALX's roadway following experiments. As discussed in previous sections, the on-board vision system is responsible for visual perception of the roadway by detection of the roadway edges. A generic vision algorithm which uses techniques such as optimized thresholding, blob detection and association, and roadway edge construction and projection has been developed and tested. Through a series of tests, the performance of the vision system was evaluated. The vision system guided ALX through a test track consisting of both straight sections and corners of varying curvatures under bright sunlight, overcast conditions, and low light intensity situations (i.e., dusk and dawn).

Figure 6-1 shows a typical input and output image for the vision algorithm. On the output image (figure 6-1b), a line approximately in the center of the blobs represents the curve fit to the right and left road edges. These images demonstrate the algorithm's ability to effectively detect the roadway edges.

Figure 6-1 (A). Typical Input image.

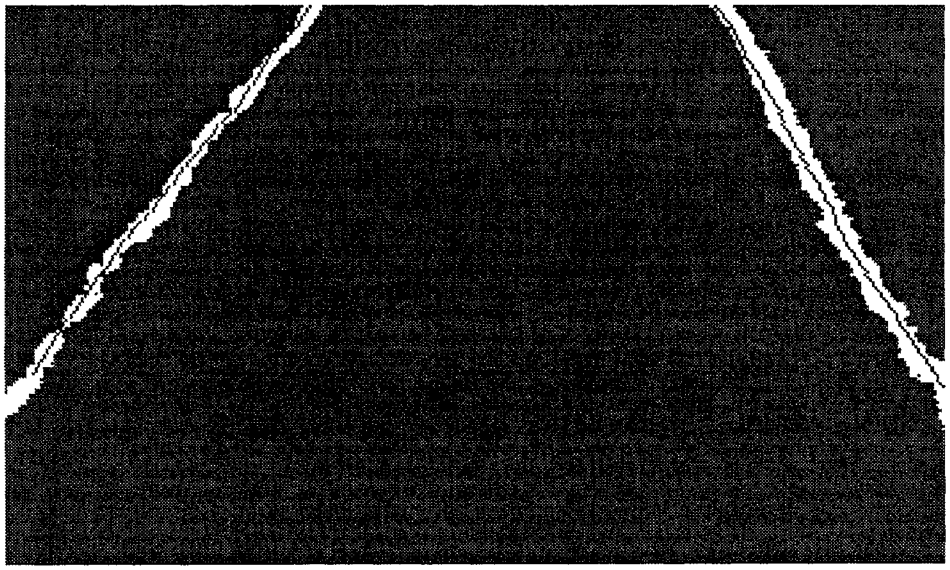
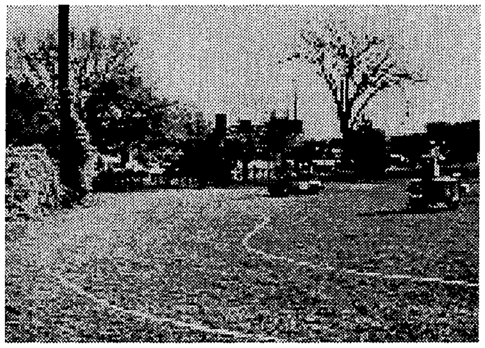
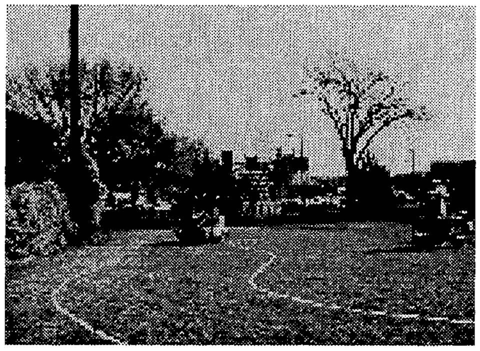
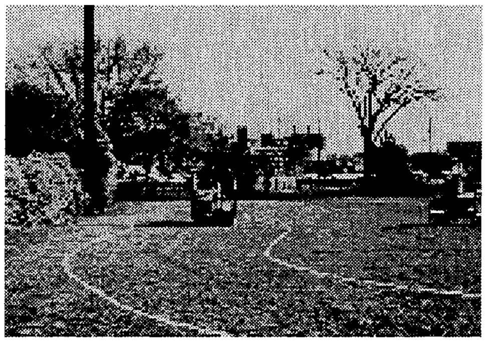


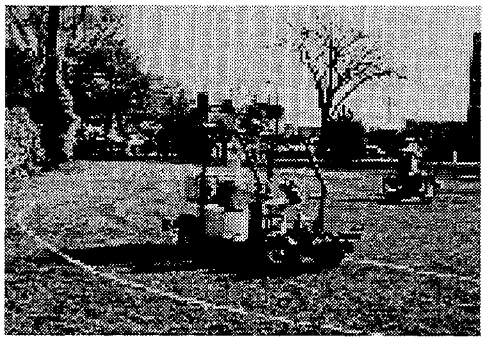
Figure 6-1 (B). Typical output image. Curve fit is thin black line.

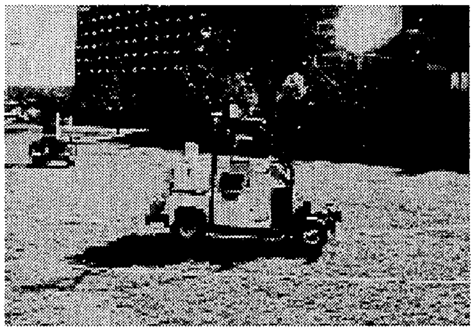
Figure 6-2 shows ALX successfully following the test track mentioned above. In our experiments, we used white color masking-tape of 3-inch width as the mark of the roadway edges. The width of the road was designed to be constant and was of 11 feet.


(a).


(b).


(c).


(d).


(e).

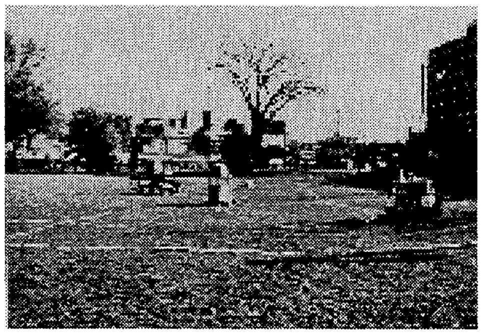

(f).

Figure 6-2. ALX is able to follow a test roadway.

As shown in Figure 6-3, a roadway turning section was designed to test ALX's ability to handle roadways containing varying curvatures. The curvature increases when entering the turning section and decreases when exiting the turning section. This kind of curvature change can be widely found in commercial freeway structures. In Figure 6-3, the largest curvature occurs at the center of the turning section, with a radius of 10 feet (3.05 meters) for the inner side of the road. The trajectory of ALX was measured manually by use of a tape-measure.

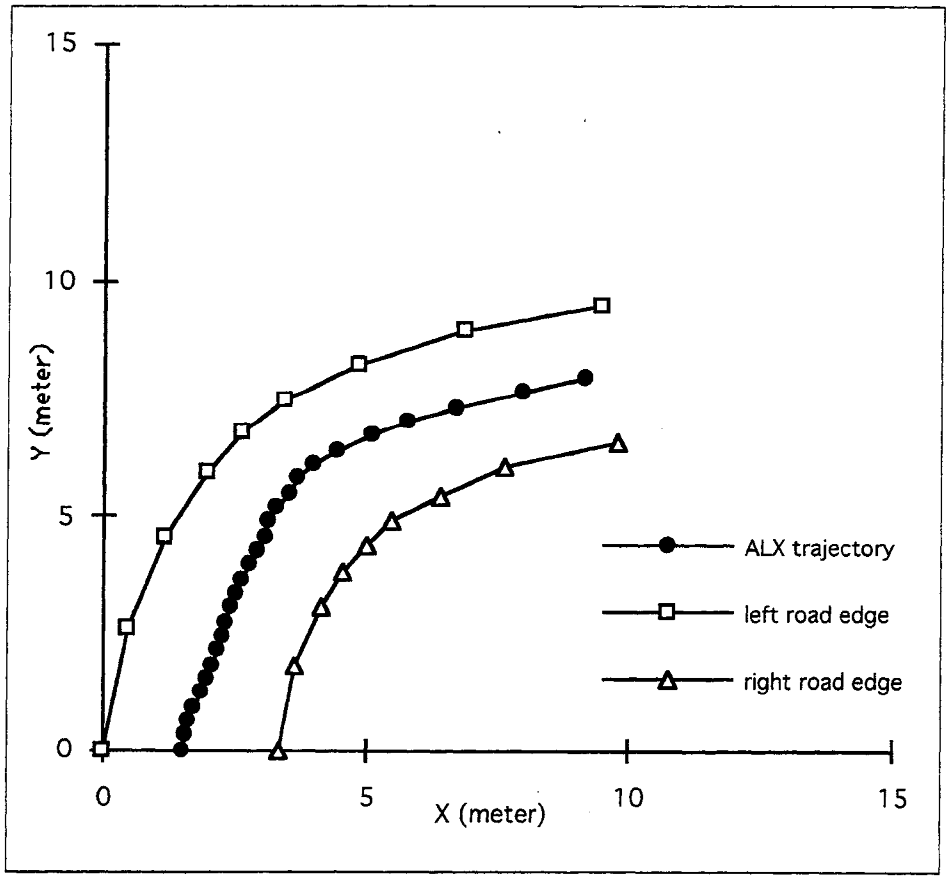


Figure 6-3. ALX follows a turning section of the testing track.

6.2 Obstacle Avoidance Test

To test ALX's obstacle detection and collision avoidance operation separately, instead of using the vision system to generate goal points, we pre-defined a series of goal points and stored them into the primary forward goal stack. (As discussed in *Section 3, Path Tracking,* any given path is defined by a series of goal points.) Various scenarios, including single obstacle either on the side of the path or totally blocking the path, and multiple obstacles at different locations, were tested.

Figure 6-4 shows ALX successfully detecting an obstacle blocking the predefined path (the primary path), backing up for more maneuvering room, generating a detouring path to go around this obstacle, resuming roadway following down the primary path and reaching the destination.

Notice that it is not necessary for ALX to back up each time it tries to go around an obstacle, provided there is enough room for the vehicle to maneuver. Since the detection range of ALX's on-board ultrasonic range sensors is very limited (only about 2.5 meters on a windy day), if the obstacle is found in the middle of the path, then there might not be enough room left for a sharp turn. In this situation, ALX has to back up an appropriate distance (usually about 0.5 meter). The backing up path is constructed by using the passed goals, based on the assumption that the traveled space is still obstacle-free, i.e., there are no dynamic obstacles.

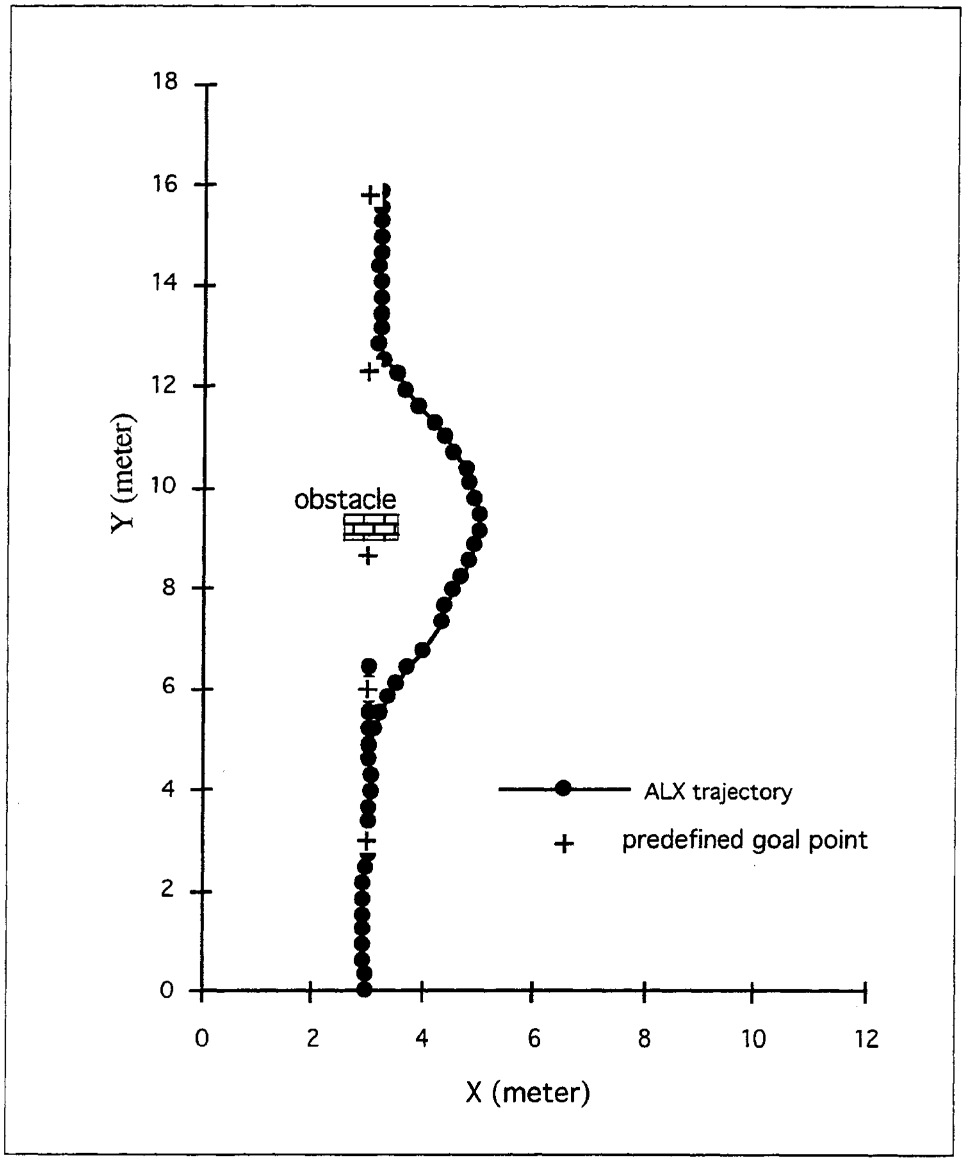


Figure 6-4. ALX obstacle avoidance operation - single obstacle. The obstacle is in the middle of the path indicated by the predefined goal points.

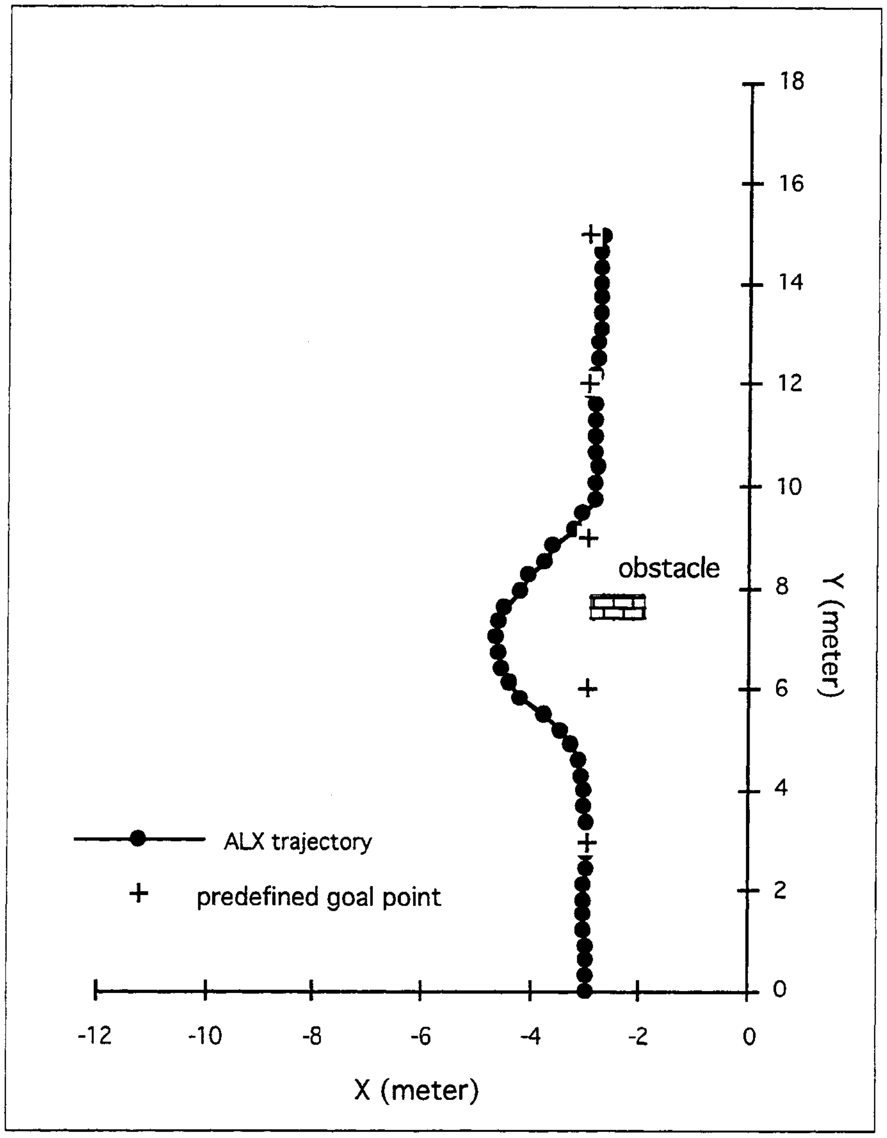


Figure 6-5. ALX obstacle avoidance operation - single obstacle. The obstacle is on the side of the path indicated by the predefined goal points.

In Figure 6-5, the obstacle is on the side of the predefined path, and there is enough free space to generate a secondary alternative path. In this situation, no backing up was performed, and ALX successfully executed the alternative path to avoid the obstacle.

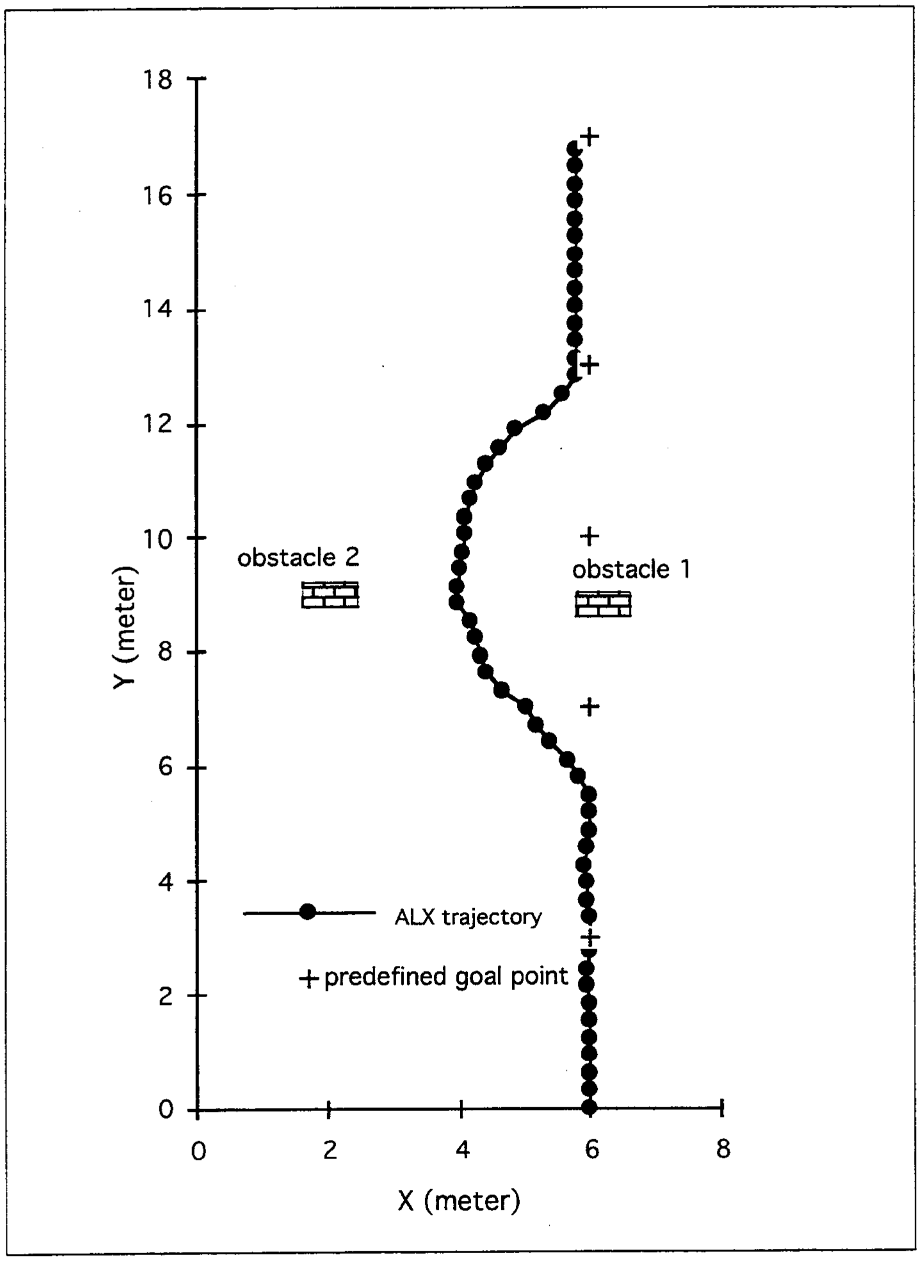


Figure 6-6. Motion of ALX in response to two obstacles, situation (a).

Two obstacles appeared in the experiment as shown in Figure 6-6. These two obstacles were about four meters apart. ALX successfully used the free space between the two obstacles to get around the first obstacle and reached the final goal point.

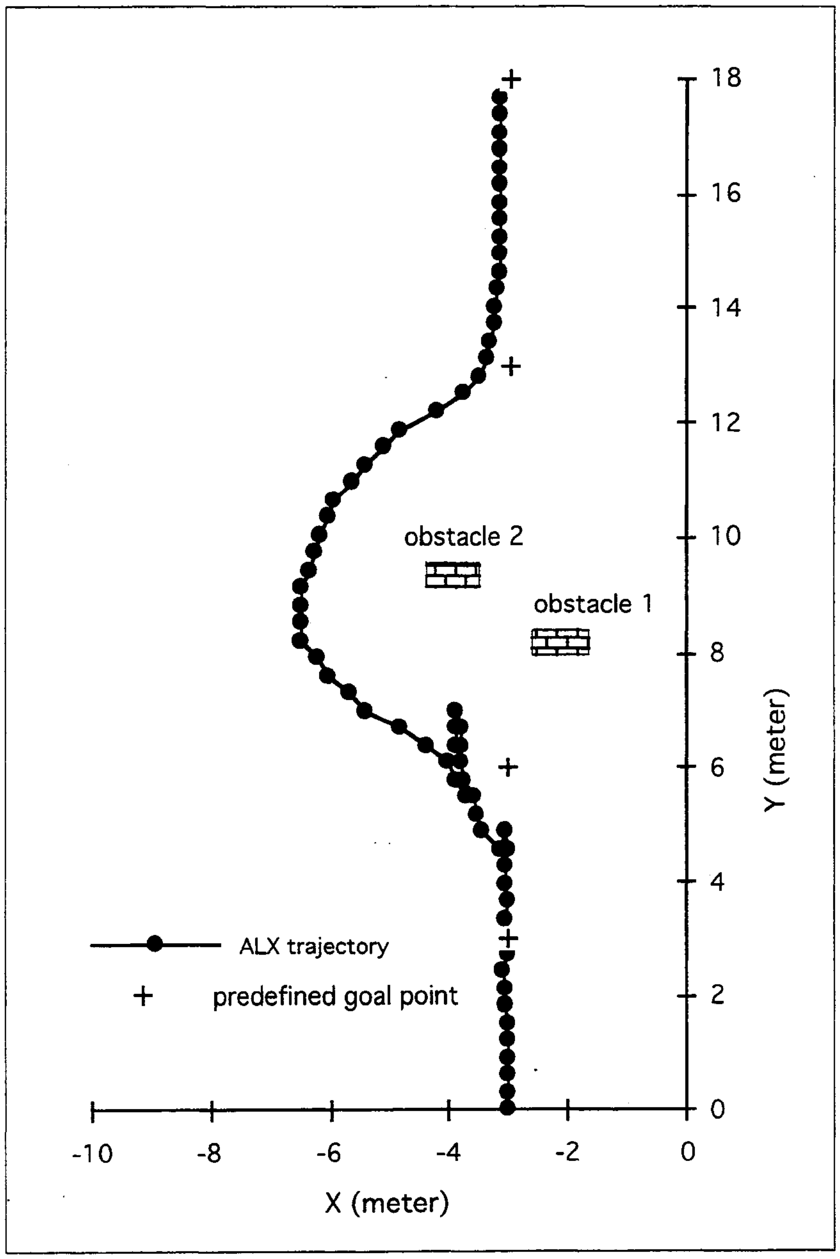


Figure 6-7. Motion of ALX in response to two obstacles, situation (b).

In Figure 6-7, two obstacles completely blocked the path. After the first obstacle was detected, an alternative path was generated. However, during the execution of this alternative path, the second obstacle was detected and determined to be collision-imminent. This forced the abortion of the alternative path execution. ALX backed up for more maneuver room and the second alternative path was generated. By using this second alternative path, ALX was able to reach the final destination.

6.3 Overall Performance Test

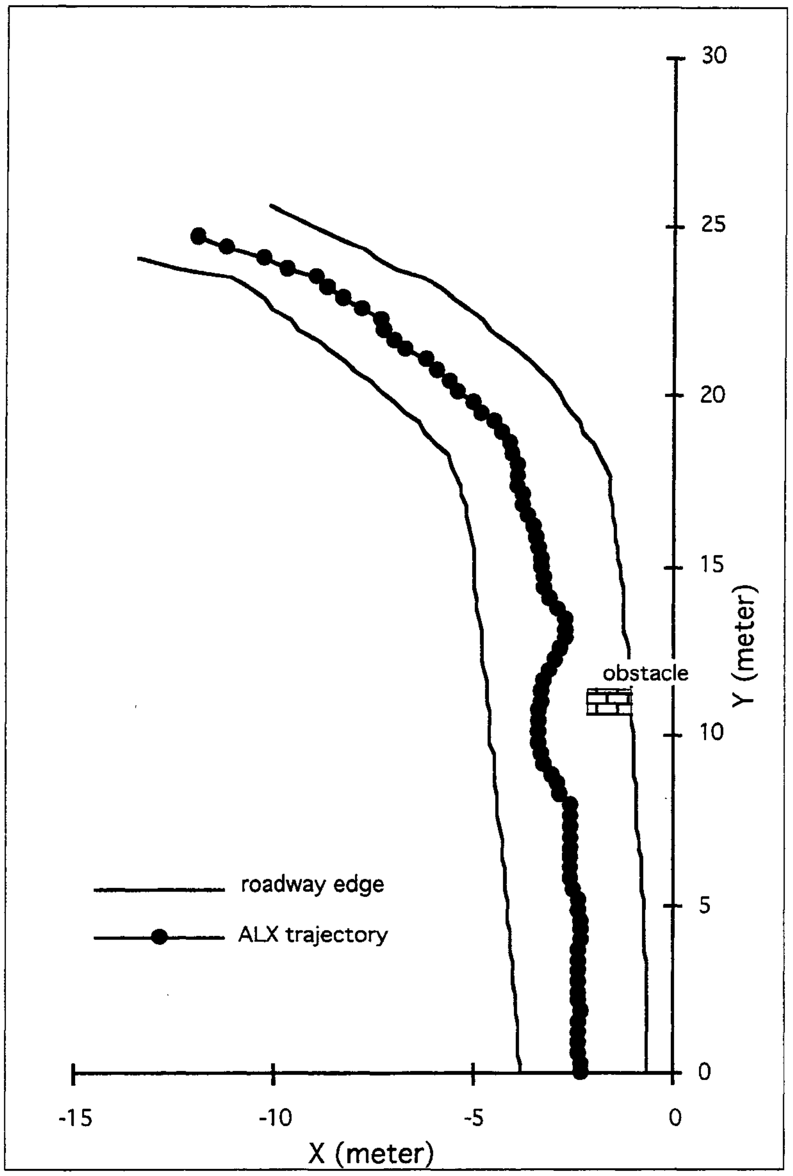


Figure 6-8. ALX performs roadway following with obstacle avoidance operation. Notice that the obstacle is on the side of the roadway.

In the third phase of our experiment, a single obstacle was placed on a roadway, as shown in Figure 6-8 and 6-9.

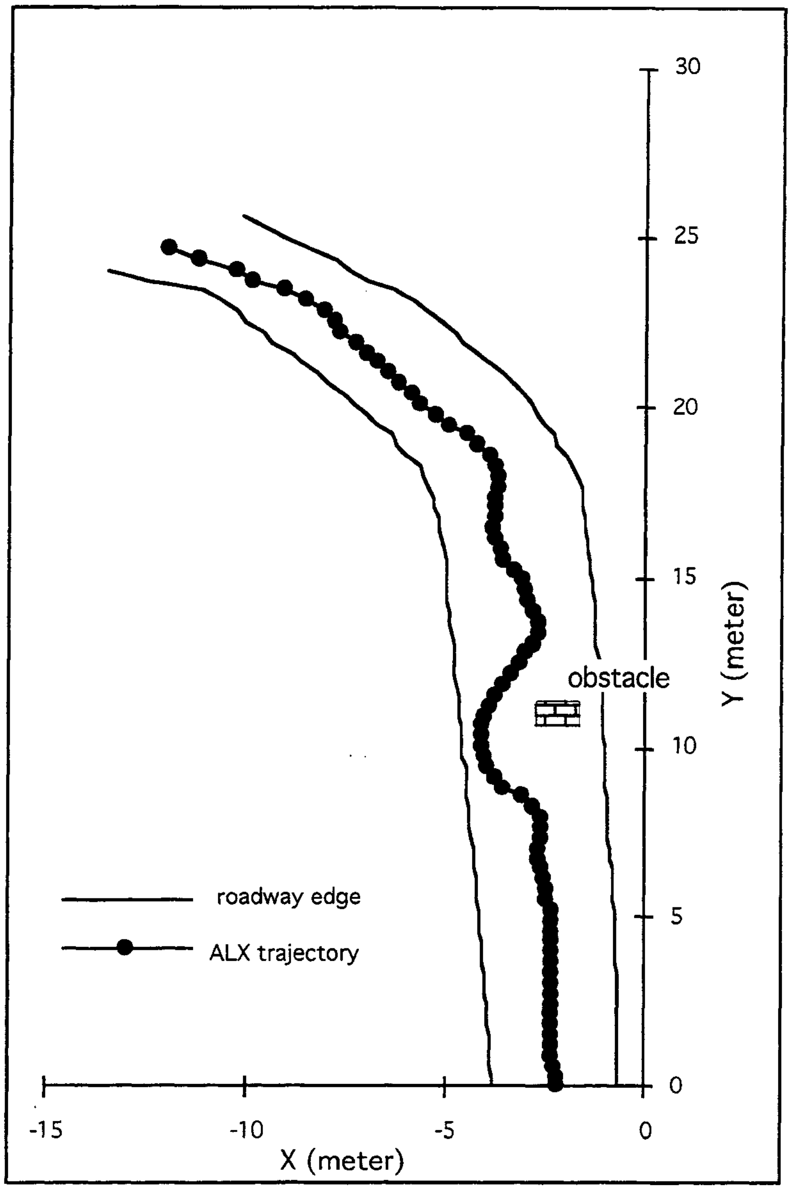


Figure 6-9. ALX performs roadway following with obstacle avoidance operation. Notice that the obstacle is near the middle of the roadway.

It is desired that the entire body of ALX should remain inside the roadway during its obstacle avoidance operation. However, if the obstacle is located near the middle of the road and there isn't sufficient room for safe maneuver, as shown in Figure 6-9, ALX will generate an alternative path that might lead to temporary lane departure. ALX is able to return to the desired lane after completing the obstacle avoidance maneuvers (Figure 6-9).

One problem we encountered during our experiment is that if an obstacle is placed at a position on the roadway that requires a very sharp vehicle maneuver, then the vision algorithm might lose track of the roadway, because the image changes dramatically between frames (currently the vision algorithm operates at roughly two frames per second). If the vision algorithm cannot recover from this situation, then the vehicle will stop and the operation is terminated. A remedy for this is to integrate the dead-reckoning information into the vision algorithm, and use this information to help the vision to anticipate where the disappeared roadway edge might emerge. This method is currently under study and will be tested in the future.

CHAPTER 7
CONCLUSIONS

7.1 System Evaluation

ALX has been successfully tested while implementing both obstacle detection and collision avoidance and vision based lateral control strategies simultaneously on a test track consisting of both straight sections and corners of varying curvatures. A number of scenarios have been tested, ranging from a single obstacle near the lane boundary to multiple obstacles which completely blocked the marked lane. ALX is able to compute and execute trajectories which require temporary lane departure if necessary, and is able to return to the desired lane after completing the obstacle avoidance maneuvers. The lighting condition for the vision system ranges from bright sunlight, overcast conditions and low light intensity situations (i.e., dusk and dawn). Given the nature of ALX, its overall performance has been satisfactory.

The following is a brief evaluation on ALX subsystems' performance and limitations.

1.) Embedded real-time control system. The embedded real-time control system has proven to be appropriate for ALX's autonomous navigation. By using the multi-tasking structure and inter-task communication mechanism, ALX is able to respond to multiple continuous and discrete events in the environment around the vehicle within time constrains. The VxWorks preemptive priority-based scheduler makes it possible to distribute computing power to different tasks according to their priorities, and guarantees that emergent events will be dealt with first. This is very difficult to implement, however, in a traditional sequential control system. Currently, this embedded system uses only one Motorola 68020 microprocessor operating at 25 MHz. This turns out to be a barely adequate computing resource and has limited the ALX's performance. For example, it usually takes about 3 to 5 seconds to compute an alternative path when ALX is obstructed. This is a relatively long period and forces the control algorithm to halt ALX during the path searching process. A smoother operation (without stopping) can be achieved if more computing power is available so that the searching period is relatively very short in terms of ALX's navigation speed.

2.) Ultrasonic range sensors. We are not satisfied with the performance of the ultrasonic range sensors. Although they usually give adequate sensing range in indoor environments, we discovered that ultrasonic range sensor are not appropriate for outdoor applications such as ALX's autonomous navigation. Besides limitations including bandwidth, signal to noise ratio, lateral spatial resolution, wind and dirt effects, etc., the ultrasonic range sensor can provide a maximum reliable detecting range of only about 3 meters or less. This short detection range always causes the vehicle to get very close to the obstacle, and as a result ALX has to backup for more room to maneuver.

3.) The vision system. The generic vision algorithm has given reasonable results. However, it is subject to sudden changes in lighting conditions. For example, if ALX moves into a direction in which the sunlight suddenly appears in the camera lens, the captured image will be very bright. It would be very difficult to calculate an appropriate threshold value which is need for further image processing. Without a proper threshold, the whole operation has to be terminated. Another issue the vision algorithm faces currently is how to recover from losing track of the roadway and how to resume roadway tracking after the roadway image has been lost. It seems that this problem can be solved only with the integration of the dead-reckoning information. Due to the present lighting conditions requirement, the vision is not able to deal with all-weather operation.

4.) Dead-reckoning system. The dead-reckoning system uses the magnetometer and wheel encoder to calculate ALX's position and orientation. Currently, the average accuracy of the positioning the dead-reckoning system provides is about 95% in the test site of 30 by 50 meters (one-way travel). However, this accuracy decays as ALX travels longer. This is due to slippage between the wheels and the road, and the accumulated error in the integration process. A system for continuously correcting dead-reckoning results can improve this situation.

5.) The obstacle avoidance algorithm. By conducting a recursive binary-tree graphic search on the virtual map, the obstacle avoidance algorithm is able to generate alternative paths successfully. However, these paths are not optimized in terms of shortest distance. Also, the path searching time is sensitive to the searching step size on the virtual map. Too small a size will increase the searching time exponentially while too big a size will cause missing possible solutions. The step size has to be determined empirically.

7.2 Recommendations

Based on the discussion above, we make the following recommendations for future research on autonomous vehicle navigation.

1.) Continue using the embedded real-time control system under the VxWorks real-time operating system; however, increase computer resources, with more powerful microprocessors and additional units.

2.) Replace the ultrasonic sensors with millimeter wave radar for obstacle detection. We have investigated millimeter wave radar technology [51] for obstacle detection. New commercially available units (e.g., from Eaton Vorad) should be evaluated. This effort will begin in the near future.

3.) Integrate the dead-reckoning information with the vision algorithm. This will help the vision algorithm to recover from loosing track of the roadway edges. For all-weather operation, consider other guidance devices rather than the CCD camera (for example, DGPS or magnetic striping).

4.) Use differential global positioning system (DGPS) to correct the dead-reckoning calculation periodically. For example, the Novatel DGPS system is specified by the vendor to provide real-time 5 Hz uninterpolated position information with an error less than 20 cm CEP. Our recommendation to evaluate such a system is presently under way.

5.) Consider Kohonen neural networks for mapping the environment around the vehicle and for vehicle control. Kohonen neural networks are capable of dealing with very complex dynamically changing environments because of their "learning" ability [46]. This effort has already begun [37]. Preliminary results on interpreting sensor data will be presented in [53].

6.) Consider alternate collision avoidance strategies such as the "virtual bumper" concept described by [48]. The ALX would serve as an excellent vehicle for testing these strategies with minimal risk to the driver or vehicle. This effort has already begun and is documented in [52].

7.) Use ALX to evaluate the use of impedance controlled steering strategies for interfacing/modulating between driver controlled steering and computer controlled steering.

7.3 Summary

As stated earlier, ALX was designed to solve a specific problem: navigating an *a priori* unknown outdoor road with or tape paint-stripe lane boundaries, under varying sunlight conditions, while avoiding obstacles of unknown size, shape, number, and placement. The intent here was to test out concepts for integrating lateral guidance and collision avoidance procedures. We do not expect that all the subsystems described in this report will be used on the truck testbed in the SAFETRUCK project. However, the software and communication architecture described here represents an early prototype that allowed us to experiment with a variety of hardware configurations, software tools, system and sensor data integration issues and interprocessor communication procedures.

ALX has been operating successfully given the nature of its task. It has proven to be a valuable tool for the preliminary investigation of vehicle sensing and control strategies at the system integration level. To date, it has facilitated the evaluation of multi-tasking and intertask communication, real-time control protocols, vision based lateral control, dead-reckoning based vehicle position estimation, and ultrasonic sensor based collision avoidance strategies. The collision avoidance strategies used here would be appropriate for maneuvering in parking lots and in freight loading areas, but not for highway applications. However, future work will investigate alternate methods for collision avoidance, especially those which would be relevant to large trucks which would have to steer around immediate collisions due to their high inertias. Higher risk strategies can be investigated on ALX without significant concern for human safety and vehicle damage.

REFERENCES

[1] H. Preston, J. Holstein, J. Otteson, and P. Hoffman, “Precursor System Analysis of Automated Highway Systems: Activity Area A - Urban and Rural AHS Analysis,” Battelle Transportation Systems, Columbus, Ohio, 1995 (FHWA Contract No. DTFH61-93-C-00195).

[2] C. Shankwitz, M. Donath, V. Morrellas, and D. Johnson, “Sensing and Control to Enhance the Safety of Heavy Vehicles,” *Proceedings of the Second World Congress on Intelligent Transport Systems*, Yokohama, Japan, Nov. 9 -11, 1995.

[3] Lumelsky, V.S. Mukhopadhyey, and K. Sun, “Dynamic Path Planning in Sensor-Based Terrain Acquisition,” IEEE Transactions on Robotics and Automation, August 1990, Vol. 6, No. 4, pp. 530-540

[4] Hiroshi Kamada, S. Naoi, and T. Gotoh, “A Compact Navigation System Using Image Processing and Fuzzy Control,” Proceedings 1990 IEEE Southeast Conference, pp. 337-342.

[5] Wiley Holcombe, et al, “Advances in Guidance Systems for Industrial Automated Guided Vehicles”, SPIE Vol. 1007 Mobile Robots III, pp. 288-297, 1988.

[6] Sven J. Dickingson and Larry S. Davis , “A Flexible Tool For Prototyping ALV Road following Algorithms”, IEEE Transactions on Robotics and Automation, Vol. 6, April 1990.

[7] B.H. Krogh and C.E. Thorpe, “Integrated path planning and dynamic steering control for autonomous vehicles, “ in Proceedings IEEE international Conference on Robotics and Automation. San Francisco, CA, April 7-10, 1986, pp. 1664-1669.

[8] Charles Thorpe, *Vision and Navigation: The Carnegie Mellon Navlab.* Klumar Academic Publishers. Boston, 1990.

[9] J. Crisman and J. Webb, "The Warp Machine on Navlab", IEEE Transactions on Pattern Analysis and Machine Intelligence," Vol. 13, No.5, May 1991, pp.451-465.

[10] Ernst Dickmanns and Birger Mysliwetz, "Recursive 3-D Road and Relative Ego-State Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence", Vol. 14, NO.2, February 1992, pp. 199-213.

[11] J. Crisman and C. Thorpe, "Color Vision for Road Following", SPIE Proceedings of Mobile Robots III, November, 1988.

[12] Karl Kluge and Charles Thorpe, "Explicit Models for Robot Road Following", Proceedings of the 1989 IEEE International Conference on Robotics and Automation, Scottsdale AZ, pp. 1148-1154.

[13] T. Fujimori and Takeo Kanade, "An Approach to Knowledge-Based Interpretation of Outdoor Natural Color Road Scenes", Chapter 4, *Vision and Navigation: The Carnegie Mellon Navlab (edited by C. Thorpe).* Klumar Academic Publishers. Boston, 1990.

[15] Sandia National Laboratories, "Techniques for Autonomous Navigation", Sandia Report, SAND92-0457, 1992.

[19] M. Beckerman and E. Oblow, “Treatment of Systematic Errors in the Processing of Wide-Angle Sonar Sensor Data for Robotic Navigation,” *IEEE Transactions on Robotics and Automation*, Vol. 6 , No. 2, April 1990, pp. 136-145.

[20] R. Kuc and V. Viard, “A Physically Based Navigation Strategy for Sonar-Guided Vehicles,” The International Journal of Robotics Research, April 1991, Vol. 10, No. 2, pp.75-87.

[21] A. Elfs, “Sonar-based real-world mapping and navigation,” IEEE Journal of Robotics and Automation, vol. RA-3, No. 3, pp.249-265, 1987

[22] H.P.Moravec, "Sensor Fusion in Certainty Grids for Mobile Robots", AI Magazine 9, No.2, 61-74 (1988).

[23] H.P.Moravec, “Certainty grids for mobile robots,” in Proc. JPL Workshop Telerobotics (Pasadena, CA), Jan. 1987.

[24] R.C.Fryxell, “Navigation planning using quadtrees,” Oak Ridge National Laboratories report, ORNL/TM-10481 (CESA-87/20), 1987.

[25] T. Lozano Perez and M.A Wesley, "An algorithm for Planning Collision-free Paths Among Polyhedral Obstacles" ACM 22, No. 10, 560-570 (1979)

[26] C.E.Thorpe, "Path Relaxation: Path planning for a mobile Robot", American Association for Artificial Intelligence Conference Austin, Texas 318-321 (1984)

[27] R.Brooks, "Solving the find-path problem by good representation of free space", IEEE Transactions System Man and Cybernetics 13, 190-197 (1983).

[28] Don Krantz, “Design Documentation Package for the ALX”, Robotics Laboratory Report, University of Minnesota, August 1993, CAMDAC Report, No. 5271-93-DKDD.

[29] Stuart Bennett, *Real-time computer control : an introduction,* Publisher: Prentice Hall, N.J., 1994.

[30] B. Schiller, “Vision Guidance Subsystem for ALX,” Robotics Laboratory Report, University of Minnesota, July 1995, CAMDAC Report, No. 5271-95-WSVG.

[31] O. Amidi, "Integrated Mobile Robot Control", M.S. Thesis, Carnegie Mellon University, May 1990.

[32] Murphy, K., " Navigation and Retro-traverse on a Remotely Operated Vehicle," IEEE Singapore International Conference on Intelligent Control and Instrumentation, Singapore, February, 1992

[33] Juberts, M., Raviv, D, Bishop, J.R. “Autonomous Road Following - A Vision-Based Approach for AVCS,” Proceedings of the International Conference on Intelligent Autonomous Systems, Pittsburgh, February 15-18, 1993

[34] Yuen-Ling Cora Lam, “ALX Kinematics Model Evaluation,” Robotics Laboratory Report, University of Minnesota, March 1994, CAMDAC Report, No. 5271-94-YLAK.

[35] M.K. Brown, “Feature Extraction Techniques for Recognizing Solid Objects with an Ultrasonic Range Sensor,” *IEEE Journal of Robotics and Automation*, Vol. RA-1, No. 4, December 1985, pp. 191-205.

[37] V. Morellas, J. Minners, and M. Donath, "Implementation of Real Time Spatial Mapping in Robotic Systems Through Self-Organizing Neural Networks," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS '95), Pittsburgh, August, 1995.

[38] C.R.Weisbin, G. de Saussure, and D. Kammer, "SELF-CONTROLLED. A real-time expert system for an autonomous mobile robot," Computers in Mechanical Engineering, pp.12-19, Sept. 1986.

[39] R.Kuc and B.Barshan, "Navigating vehicles through an unstructured environment with sonar," in Proc. IEEE Int. Conf. Robotics Automation (Scottsdale, AZ, May 14-19, 1989), pp.1422-1426.

[40] J.L. Crowley, "World modeling and position estimation for a mobile robot using ultrasonic ranging", in Proc. IEEE Int. Conf. Robotics Automation (Scottsdale, AZ, May 14-19, 1989), pp. 674-680.

[41] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in Proceedings IEEE International Conference on Robotics and Automation. St. Louis, MO, Mar. 25-28, 1985, pp. 500-505.

[42] Krogh, B. "A generalized potential field approach to obstacle avoidance control," presented at International Robotics Research Conference, Bethlehem PA, Aug. 1985.

[43] Y. Koren and J. Borenstein, "Real-time obstacle avoidance for fast mobile robots," IEEE Trans. Systems Man Cybern., vol. 19, no. 5, pp.1179-1187, Sept./Oct. 1989.

[44] J.Borenstein and Y. Koren, “The vector field histogram - fast obstacle avoidance for mobile robots”, IEEE Trans. on Robotics and automation, Vol. 7. No.3, June, 1991.

[45] Curt Olson, “Constrained Obstacle Detection and Avoidance”, Robotics Laboratory Report, University of Minnesota, March 1994, CAMDAC Report, No. 5271-94-COCO.

[46] Vassilios Morrelas, "A Neuro Computational Mobile Robot Architecture for Mapping Localization and Navigation", Ph.D Thesis, University of Minnesota, June 1995.

[47] M. Hennessey, C. Shankwitz, and M. Donath, “Sensor Based ‘Virtual Bumpers’ for Collision Avoidance: Configuration Issues,” Proceedings of the Collision Avoidance and Automated Traffic Management Sensors Conference, Philadelphia, PA, SPIE (Conf. No. 2592), October 1995.

[48] Datacube, Inc., MaxVideo 200 Hardware Reference Manual, Document No. HW600-1.0, March 1993.

[49] Datacube, Inc., ImageFlow System Manual Update Package, Document No. IF102-5.1, July 1994.

[50] Dennis J. Wenzel, Steven B. Seida, "High Speed Extraction of Line Segment Features", Jan 1993 SPIE, vol. 2064, pp. 47-58.

[51] Shankwitz, C. and Donath, M., “MIMIC Sensor technology for highway vehicle applications: Potential and Challenges for the future,” MnDOT (report # MN/RC - 95/10), March 1995.

[52] Schiller, B. et al. (in press), "Vehicle guidance architecture for combined lane tracking and obstacle avoidance." In D. Kortenkamp, R.P. Bonasso, and R. Murphy (Eds.) "Artificial intelligence and mobile robots." MIT/AAAI Press, Cambridge, MA.

[53] Garlich-Miller, M and Donath, M., "A connectionist approach to the fusion of three-dimensional, sparse, unordered sensor data," Proceedings of the Japan/USA Symposium on Flexible Automation, v. 1, pp. 465-473, ASME, New York, 1996.

APPENDIX A

DEAD-RECKONING SYSTEM

APPENDIX A
DEAD-RECKONING SYSTEM

A.1 Introduction

The dead-reckoning system calculates ALX's position in Cartesian coördinate space based on the inputs from a suite of dead-reckoning sensors. Other entities sensed by ALX (roadway edges, obstacles, etc.) are then referenced by using ALX's current position and orientation, i.e., the results of the dead-reckoning computation.

The dead-reckoning system consists of a set of dead-reckoning sensors, which include one three-axis fluxgate magnetometer, one angular rate sensor, two clinometers, and one odometer (rear wheel encoder). The magnetometer, clinometers, and angular rate sensor are co-located in a single package - the dead-reckoning sensor head (DRS head).

Mounted on the ALX roll cage, the DRS head is mounted on an aluminum post on the passenger side of ALX to help isolate it from the magnetic disturbances caused by ferromagnetic structure of the vehicle and its electrical components (motors, batteries, power cables, etc.).

The clinometer and angular rate sensor are not currently being used in ALX's dead-reckoning system. The clinometer has been calibrated and can be used if ALX is to navigate on non-planar surfaces.

The three-axis fluxgate magnetometer (Watson Industries, Inc.) is used to determine the magnetic north vector. The output for each axis is a sinusoidal voltage signal with the correspondence shown in Figure A-1.

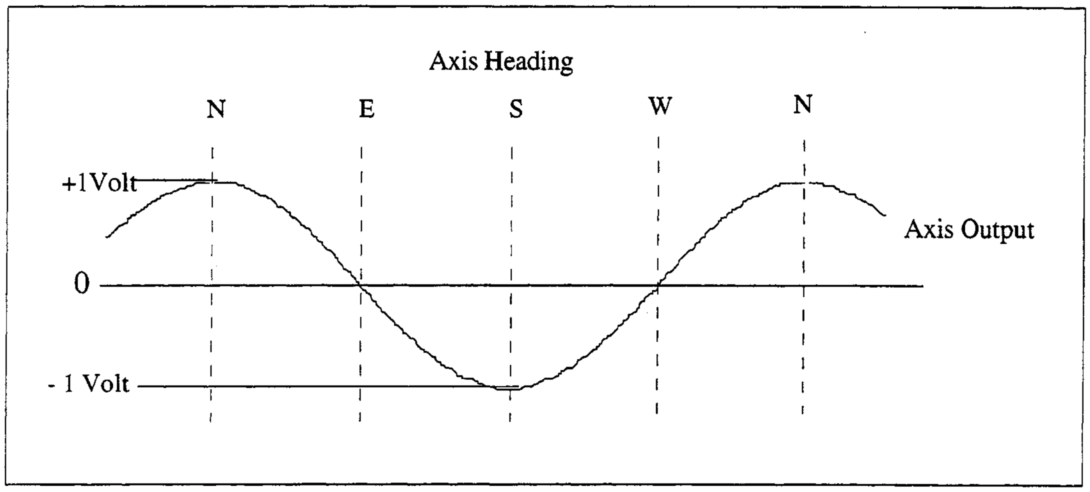


Figure A-1 Theoretical output of one axis of the fluxgate magnetometer. The output of the other axis is also a sinusoidal signal with the same amplitude but a 90• phase shift.

The fluxgate magnetometer employs Faraday's law of induction combined with the hysteresis exhibited by all ferromagnetic material. The ferromagnetic material is wound with two coils. A sinusoidal current is applied to one of the coils, saturating the core material each half cycle. With no external field, the average signal sensed by the other coil is zero. However, when an external field is present, the hysteresis loop is shifted and becomes asymmetric at about H=0, and a flux rich in the second harmonic of the excitation frequency is sensed by the second coil. In our application, the Earth's magnetic field will add to or subtract from the field produced by the reference oscillator to the phase of the detected field. The outputs of the two axes can be used to determine north using the relationship:

$$Vn = \arctan(Vy/Vx) \qquad \text{(A-1)}$$

There is only one basic design requirement for the dead-reckoning system: it must have the ability to determine the location of the vehicle with respect to the pre-established initial real-world coordinates, i.e., it must be able to determine the real-world location of the vehicle at any moment. From this point on, "location" will be defined as the orientation of the vehicle and its location in real world coordinates (easting and northing as referenced to the initial real world coordinates).

There are a few complications that arise when implementing this sensor technology on the ALX. The greatest of these is the magnetic field present within the environment of the vehicle. These magnetic anomalies affect the functionality of the highly sensitive fluxgate magnetometer, thus causing a biased sinusoidal output signal. Due to this and the fact that the Earth's magnetic field varies at different locations, a calibration scheme must be performed at each locale before vehicle operation. This procedure is a major design consideration of the dead reckoning system and is described below.

A.2 Magnetometer Calibration

Assume that we have a set of data points (x_i,y_i), where x_i is the heading angle calculated by using the magnetometer output signals (see Equation A-1), and y_i is the "true" heading angle. Since x_i and y_i are not necessarily the same because of the bias in the magnetometer and fluctuation in the magnetic field, a model is needed to relate x_i and y_i accurately so that later during ALX's navigation, we can use the angles based on the magnetometer readings as input to this model in order to obtain accurate heading angles as output. The model we use a polynomial function of x as follows:

$$y(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_M x^{M-1} \qquad \text{(A-2)}$$

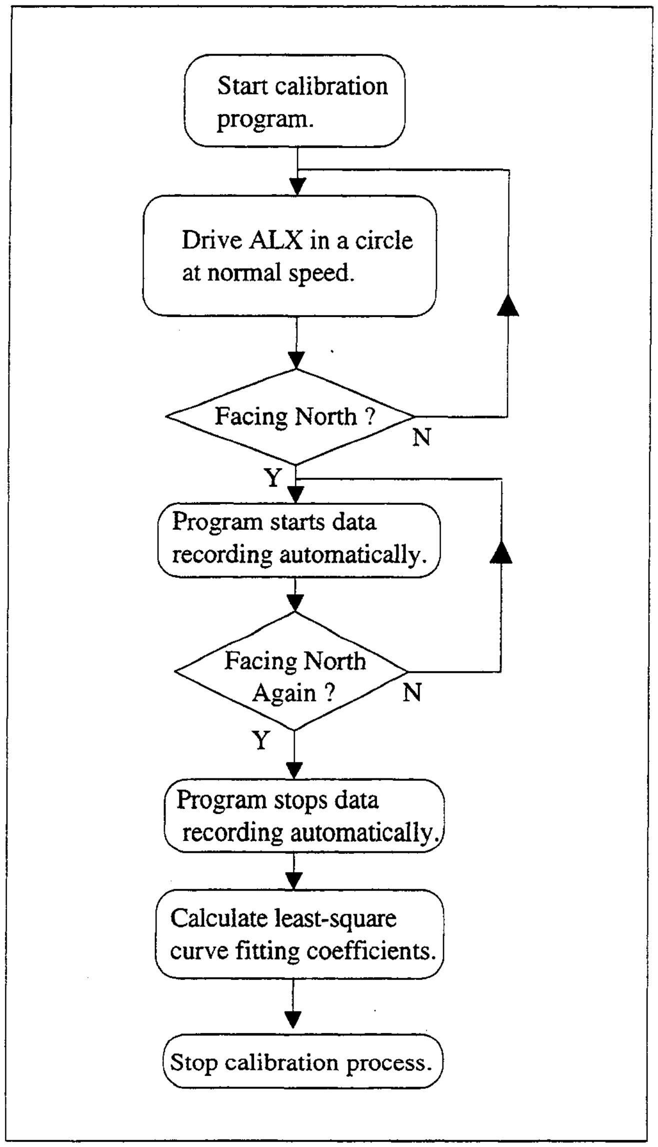
The least-square method is used to optimize the parameter set $a_{1 \ldots} a_M$ so that the merit function

$$\chi^2 = \sum_{i=1}^{N} \left[y_i - \sum_{k=1}^{M} a_k x_i^{M-1} \right]^2 \quad \text{(N is the number of data points)} \qquad \text{(A-3)}$$

is minimized[1] .

1 A detailed discussion on the least-square fitting method can be found in *Numerical recipes in C : The Art of Scientific Computing* by William H. Press, 2nd edition, Cambridge University Press, 1994.

In order to calculate the parameter set $a_1 \ldots a_M$ in Equation (A-2), a set of data point (x_i, y_i) is needed. To obtain this, a calibration procedure for the magnetometer and several supporting computer programs were developed. Figure A-2 is the flowchart of the calibration procedure.



Start calibration program.
Drive ALX in a circle at normal speed.
Facing North ?
N
Y
Program starts data recording automatically.
Facing North Again ?
N
Y
Program stops data recording automatically.
Calculate least-square curve fitting coefficients.
Stop calibration process.

Figure A-2 Flowchart of the magnetometer calibration process.

To implement the calibration process, start the auto-calibration program and then drive ALX in a circle either in the clockwise or counterclockwise direction. The program monitors ALX's heading based on the outputs of the magnetometer. When ALX faces North, the program will automatically start the data sampling and recording process. The following data will be recorded at a fixed sampling frequency (20Hz): magnetomenter X axis reading Vx(i), magnetometer Y axis reading Vy(i), and the rear wheel encoder reading e(i). After ALX has completed a circular trajectory and faces North again, the calibration program stops the sampling and recording process. The recorded data is then processed to generate the data set (x_i,y_i) needed for least-square curve fitting. Refer to Figure A-3. For each sampled time instance i, by using Equation (A-1), the heading of ALX based on the magnetometer reading is calculated as $\phi(i)=$ atan2[Vy(i)/Vx(i)]. At the same time, the "true" heading angle relative to North for this location can be calculated by using the encoder information by using the process described as follows.

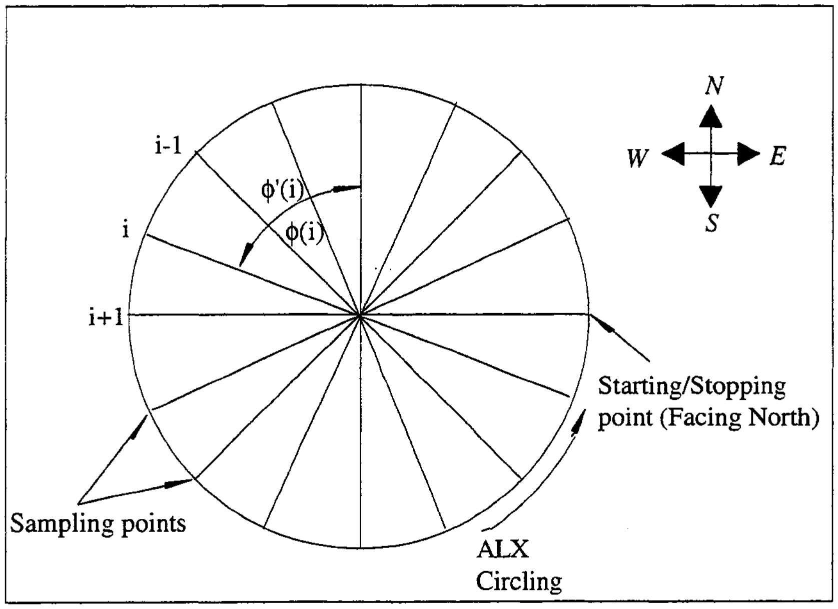


Figure A-3 Generate data for least-square curve fitting. $\phi(i)$ is calculated by using the magnetometer readings and $\phi'(i)$ is calculated by using encoder readings.

Assume ALX has moved in a perfect circle (in practice, the steering wheel of ALX is locked to one end to ensure no steering change during the circular movement). The circumference of this circle is L which can be calculated by using encoder readings at the starting and stopping point. Then the true angle $\phi'(i)$ is: $\phi'(i) = 2\pi \cdot e(i)/L$. The pair of values $(\phi'(i), \phi(i))$ then form the data set (x_i,y_i). By calling a subroutine in the calibration program, the parameter set $a1...a_M$ is determined using the least-square fitting procedure.

The major error sources for calibrating the "true" heading angles are as follows. (1) The starting and stopping points may not coincide in which case a perfect circle model cannot be obtained. In this case, there will be an error in the true heading angle calculation; (2), The true heading calculation relies on the encoder readings. However, if the ground used for testing is horizontal but not flat then the encoder readings will not match with the distance that ALX could have traveled if it were moving flat surface as in the model. A more accurate calibration method can be obtained by mounting the magnetometer to a turn table instead of using the encoder readings from the vehicle wheels.

A.3 Dead-Reckoning Position Calculation

The ALX dead-reckoning system assumes that ALX is traveling on a planar surface, which can be conveniently represented via a Cartesian coordinate system. When the system is initialized, a known starting position is required from which all further measurements are calculated. Once initialized with this known starting point, the dead-reckoning system measures the vehicle's heading relative to north and the distance traveled.

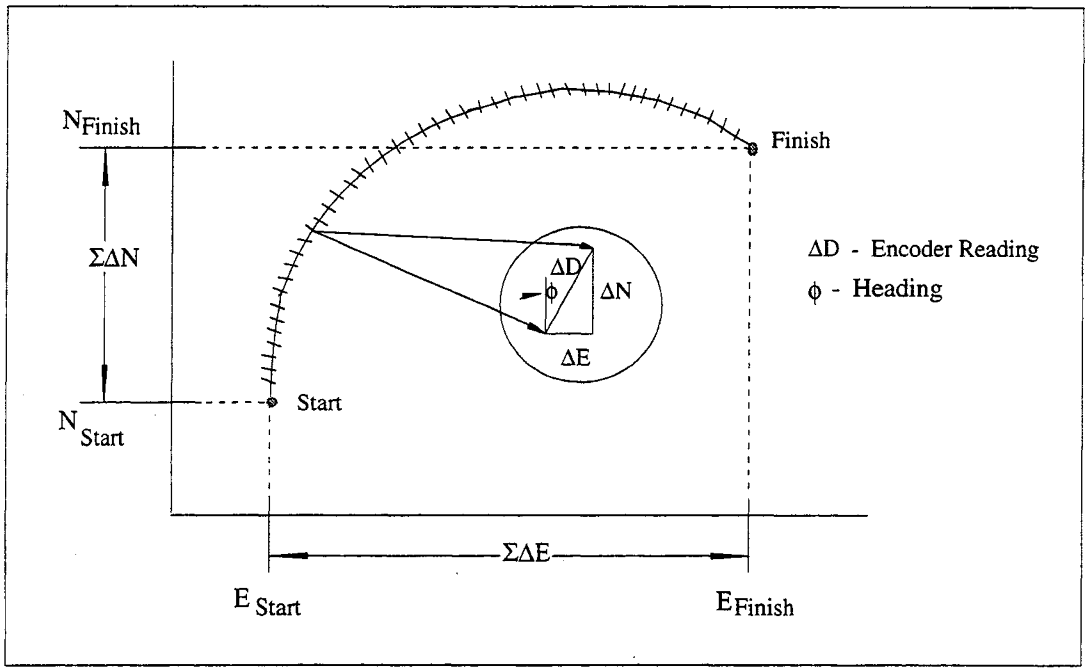


Figure A-4 Dead-reckoning position calculation.

Operating at 20Hz, the system periodically updates its position by resolving the distance traveled into the east, E and north, N, components and adding these to the last known position by using the following equations:

$$\begin{aligned} E_i &= E_{start} + \Sigma\Delta E \\ &= E_{start} + \Delta D_1 \sin\phi_1 + \Delta D_2 \sin\phi_2 + \Delta D_3 \sin\phi_3 + \ldots + \Delta D_i \sin\phi_i \end{aligned} \quad \text{(A4)}$$

$$\begin{aligned} N_i &= N_{start} + \Sigma\Delta N \\ &= N_{start} + \Delta D_1 \cos\phi_1 + \Delta D2 \cos\phi_2 + \Delta D3 \cos\phi_3 + \ldots + \Delta D_i \cos\phi_i \end{aligned} \quad \text{(A-5)}$$

In the ALX dead reckoning navigation system, measurement of the vehicle heading and the distance traveled are used to compute increments (Δ) of eastings and northings from a known starting location. The fluxgate magnetometer provides the heading reference (ϕ), and the rear-wheel encoder is used for determining the distance traveled (ΔD) in each sampling period. As shown in Figure A-4, the Δ east and Δ north values are computed as the distance traveled times the sine or cosine of the heading respectively.

APPENDIX B
VISION DATA TRANSFORMATION

APPENDIX B
VISION DATA TRANSFORMATION

The ALX on-board image processing unit processes snap-shots of the roadway and extracts the roadway edges. The result is then passed to ALX system control processor in the form of a set of data points which represent the roadway edges. These points are in the image coordinate frame and therefore need to be transformed into the world coordinate frame. Since we assume that ALX navigates on a true planar surface, the image points only need to be projected onto the ground plane instead of into 3-D space. This greatly simplifies the coordinates transformation process. (The equations for the vision data transformation in this section were first derived by Craig Shankwitz at the Robotics Labotory, University of Minnesota.)

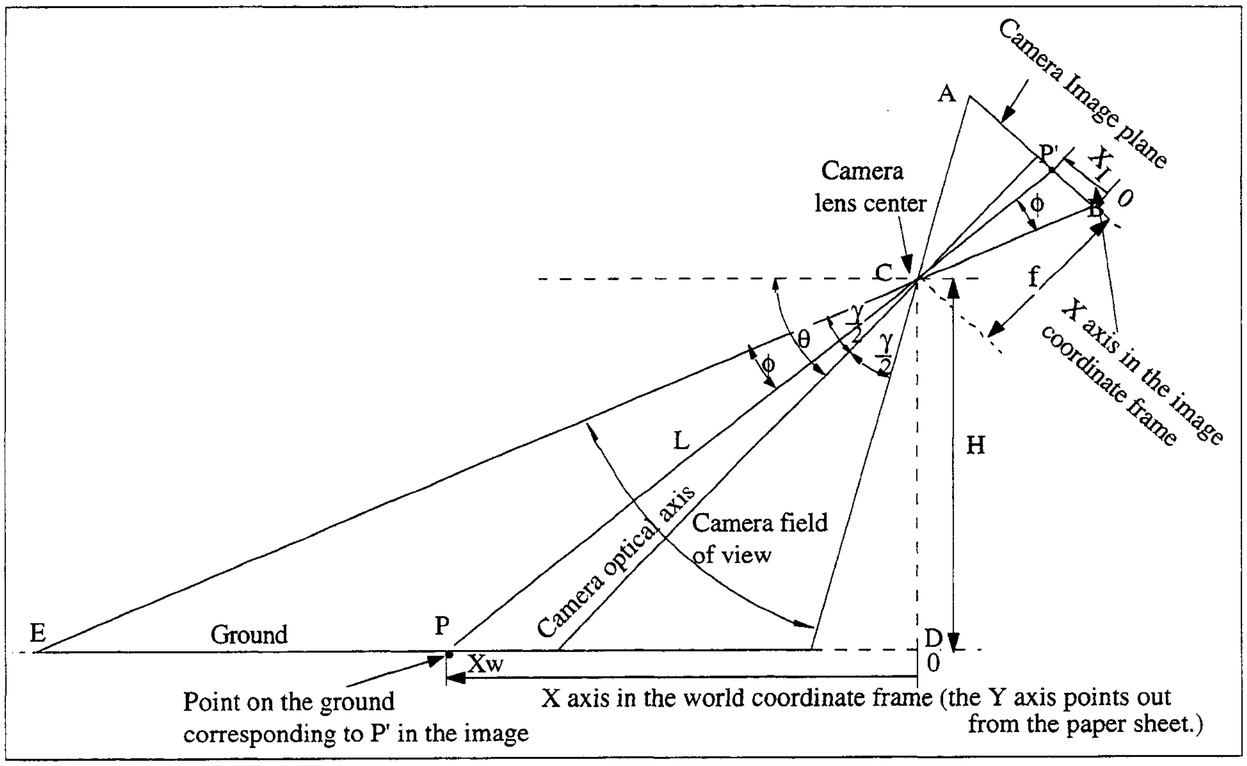

Camera Image plane
A
P'
X_I
0
B
Camera lens center
C
ϕ
f
θ
$\frac{\gamma}{2}$
$\frac{\gamma}{2}$
ϕ
X axis in the image coordinate frame
L
H
Camera optical axis
Camera field of view
E
Ground
P
D
Xw
0
Point on the ground corresponding to P' in the image
X axis in the world coordinate frame (the Y axis points out from the paper sheet.)

Figure B-1 Camera Model for Image Data Transformation (side view).

In the following, we illustrate how to project a point P' in the image coordinate frame to point P on the ground, i.e., how to transform the coordinates P'(XI, YI) in the image frame into P(Xw, Yw) in the world coordinate frame. The camera is mounted at the center of the vehicle that is also the origin of the world coordinate frame in which the X axis is point forward and the Y axis is pointing to the right side of the vehicle. The origin of the image is at its upper left corner with X axis pointing down and Y axis pointing to the right (note that in conventional image processing notation X and Y axis in the image plane are defined oppositely).

A linear camera lens model is used and illustrated in Figure B-1. The notations in Figure B-1 are as follows:

θ:	Pitch angle of the camera optical axis measured from the horizontal plane (units: degree).
γ:	Camera field of view as indicated by an angle measured about the X axis of the image coordinate frame (unit: degree).
η:	Camera field of view as indicated by an angle measured about the Y axis of the image coordinate frame (unit: degree).
H:	Camera mounting height measured by using tape measure from ground (unit: meter).
f:	Camera focal length in pixel unit. A method for determining f is presented later in this section.

Based on the trigonometric relations in the upper triangle ABC, the following equation applies:

$$\frac{X_I}{\sin(\phi)} = \frac{f/\cos(\gamma/2)}{\sin(180^\circ - (90^\circ - \gamma/2) - \phi)} = \frac{f/\cos(\gamma/2)}{\cos(\gamma/2 - \phi)} \qquad \text{(B-1)}$$

Since

$$\cos(\alpha - \beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta \qquad \text{(B-2)}$$

then Equation (B-1) can be written as:

$$\frac{X_I}{\sin(\phi)} = \frac{f / \cos(\gamma / 2)}{\cos(\gamma / 2)\cos\phi + \sin(\gamma / 2)\sin\phi} \qquad \text{(B-3)}$$

This results in:

$$\tan\phi = \frac{\cos(\gamma / 2)}{\frac{f}{X_I \cos(\gamma / 2)} - \sin(\gamma / 2)} \qquad \text{(B-4)}$$

Therefore,

$$\phi = \tan^{-1} \frac{\cos(\gamma / 2)}{\frac{f}{X_I \cos(\gamma / 2)} - \sin(\gamma / 2)} \qquad \text{(B-5)}$$

To calculate the focal length f (in pixel values) of the camera lens, an experiment can be conducted as illustrated in Figure B-2.

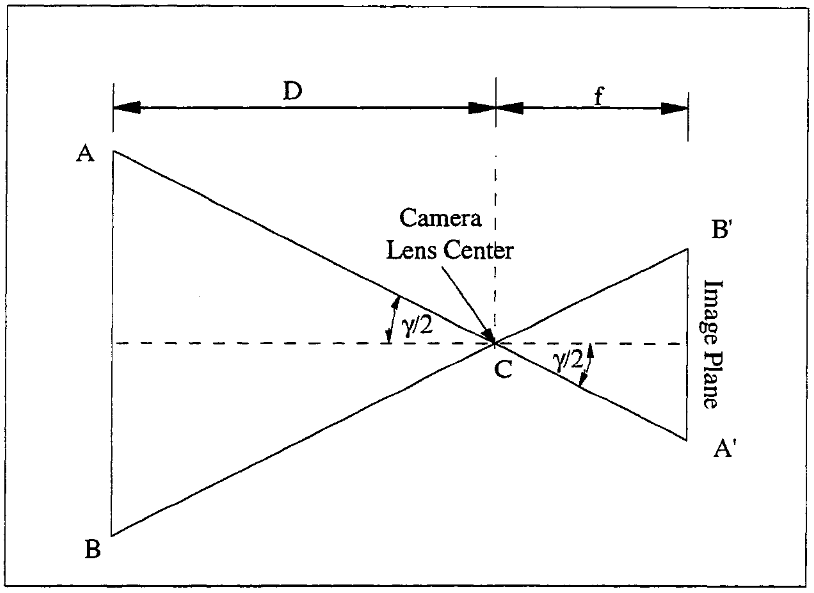


Figure B-2 Calculation of the focal length of the camera.

As shown in Figure B-2, the coordinates of A and B, and the distance from segment AB to the center of the camera D can be measured. The coordinates of A' and B' (the image equivalents of point A and B, respectively) are determined from the image. Thus from the trigonometric relationship:

$$\frac{Y_w(A) - Y_w(B)}{D} = \frac{Y_I(A) - Y_I(B)}{f} \qquad \text{(B-6)}$$

the camera's focal length can be calculated as:

$$f = \frac{Y_I(A) - Y_I(B)}{Y_w(A) - Y_w(B)} D \text{ (pixel)} \qquad \text{(B-7)}$$

In the lower triangle CDP as shown in Figure B-1, the following equations applies,

$$X_w = H \tan(\theta - \gamma / 2 + \tan^{-1} \frac{\cos(\gamma / 2)}{f / X_I \cos(\gamma / 2) - \sin(\gamma / 2)}) \qquad \text{(B-8)}$$

By substituting ϕ from Equation B-5, then equation B-8 yields the formula for Xw as:

$$X_w = H \tan(\theta - \gamma / 2 + \tan^{-1} \frac{\cos(\gamma / 2)}{f / X_I \cos(\gamma / 2) - \sin(\gamma / 2)}) \qquad \text{(B-9)}$$

The procedure for calculating Yw given (XI, YI) follows. Figure B-3 is a top view of the plane which goes through PP' in Figure B-1 and is perpendicular to the side view of Figure B-1.

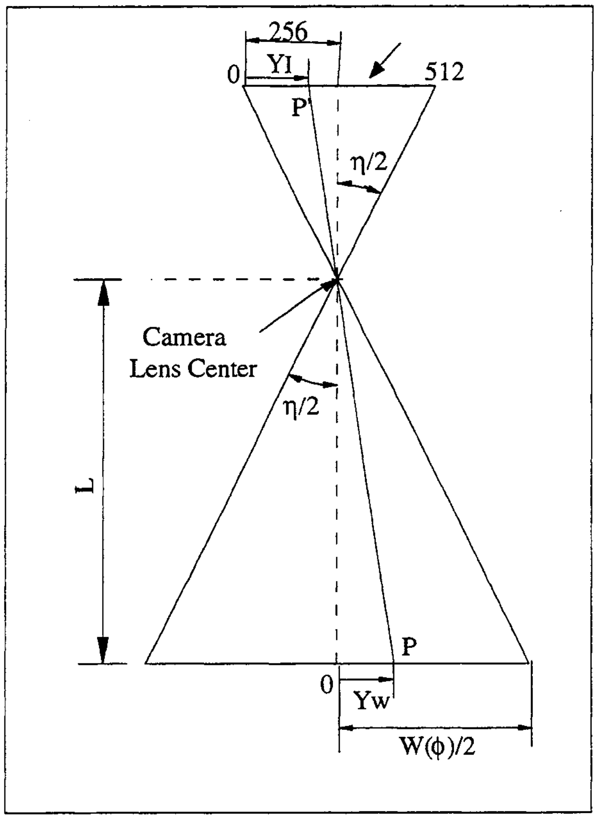


Figure B-3 Calculation of Yw (front view of the camera mounted on ALX.)

Referring to Figure B-1, the following equation applies:

$$L = \frac{H}{\sin(\theta - \gamma / 2 + \phi)} \qquad \text{(B-10)}$$

Since the X axis of the camera passes through the vehicle center, then half of the total field of view at angle ϕ in Figure B-3, is:

$$W(\phi) = 2L\tan(\frac{\eta}{2}) = 2\frac{H}{\sin(\theta - \gamma / 2 + \phi)}\tan(\frac{\eta}{2}) \qquad \text{(B-11)}$$

or,

$$\frac{W(\phi)}{2} = \frac{H}{\sin(\theta - \gamma / 2 + \phi)}\tan(\frac{\eta}{2}) \qquad \text{(B-12)}$$

The range of YI in the image frame is from 0 to 512. From the trigonometric relationship as shown in Figure B-3, we can calculate:

$$Y_w = \frac{Y_I - 256}{256}\frac{W(\phi)}{2} \qquad \text{(B-13)}$$

Substituting for W(ϕ) with Equation B-12, we then have the equation to calculate Yw as,

$$Y_w = \frac{Y_I - 256}{256}\frac{H}{\sin(\theta - \gamma / 2 + \phi)}\tan(\frac{\eta}{2}) \qquad \text{(B-14)}$$

where

$$\phi = \tan^{-1}\frac{\cos(\gamma / 2)}{\frac{f}{X_I\cos(\gamma / 2)} - \sin(\gamma / 2)} \qquad \text{(B-5)}$$

The vision data transformation equations discussed in this section were used in the initial research work done in the ALX project and satisfactory computation results have been obtained. There are some limitations of these equations. For example, one assumption used to derive these equations is the linearity of the lens. However, the linearity of the lens used on ALX is not perfectly linear, especially at the part close to the edge of the lens. This results in a less accurate transformation for the pixel points which are far away from the center of the image. We recommended a further evaluation of these equations, including a study on their limitations and the sensitivity of various parameters, etc.

APPENDIX C

VISION ALGORITHM FOR ROAD FOLLOWING

APPENDIX C
VISION ALGORITHM FOR ROAD FOLLOWING

C.1 Tuning and Running Blob Analysis

As stated in section 3, blob analysis has one major shortcoming. Blob analysis is only as good as the threshold selected for the image. For instance, if the threshold selected is too low, there will be increased noise in the image which will increase required computation and processing time. Since the vehicle is moving, the image brightness and amount of specular reflections will vary. These issues are compensated for partially by using a polarizing filter which reduces glare and having an auto-iris on the camera. Clearly the threshold selection process has to account for these changes in the image. This section discusses how the thresholding operation works as well as the process for tuning the blob analysis to work on various roadway images.

The thresholding selection is very straight-forward. The algorithm first determines the pixel intensity with the highest occurrence (P_{mean}) using the histogram data. The threshold value is then set to P_{mean} plus an offset input by the user (O_{input}). This is a very simple approach that has proven to have effective results.

Tuning the blob analysis consists of selecting an appropriate O_{input}. To aid in selecting O_{input} a "live histogramming" program has been created. This program is a live image of the roadway as seen by the ALX with a current histogram displayed in the corner of the screen. This program also accepts an O_{input} and prints out the current value of P_{mean}. Pixels with intensity values greater than the threshold (Pmean + O_{input}) are displayed in red on the screen and on the histogram. This program is used to experiment with values of O_{input} until the lines of the image are completely red with as little noise as possible.

Running the blob analysis requires using an appropriate account on the datacube. To run the program, type the following command:

example Pmean_est O_{input} LowestThreshold

For the above command, example is the executable name, Pmean_est is a best guess of what P_{mean} is (use P_{mean} from the live histogram program), O_{input} is the user input offset (tuned from the live histogram program). LowestThreshold is the lowest threshold value that the program will be able to perform (the program will be able to perform thresholds from LowestThreshold to LowestThreshold+128 in steps of 1 so never set this parameter greater than 127). Some good (typical) values for the above parameters are:

```
example 72 35 35
```

After starting the program view the output on the monitor to verify that the noise level and the line fit is acceptable.

APPENDIX D

PROGRAM DESCRIPTIONS

APPENDIX D
PROGRAM DESCRIPTIONS

The real-time control software for ALX consists of a set of C programs. These programs and their functionality together with function modules in each program are listed in Table D-1 in alphabetical order.

Table D-1 Program List

Program	Function	Module Included
adc.c	Analog-to-digital converter low-level driver. Reads the ADC, returns the value as a 'double', units = volts. Works with o+r VMIO 21/22 module.	*adc ( )*
cali.c	Dead-reckoning sensor calibration software. Records data from the calibration process and calculates coefficients for curve-fitting.	*vector( ), matrix( ), ivector( ), free_vector( ), free_matrix( ), nrerror( ), gaussj( ), covsrt( ), calread( ), calreceive ( ), record ( ), fit ( ), mrqcof ( )*
camera.c	Verifies the coordinates transformation from ground to camera screen image. Used only in the lab, not part of the control program of ALX.	*main ( )*
collect.c	Collect experimental data and record them on the ramdrive.	*init_drive( ), collect ( ), startc( ), stopc( )*
comm.c	Defines inter-task communication protocol.	*send_a_message ( ), get_a_message( )*

Table D-1 Program List (continued)

Program	Function	Module Included
dead_reck.c	Calculates dead_reckoning information, i.e., current easting, northing and orientation of ALX	*drinit( ), dr_read( ), dr_receive( ), where( )*
encoder.c	Reads right rear wheel encoder values using VMIO-27 encoder board.	*encoder ( )*
geom.c	Handles geometric calculation of the virtual map (e.g., distance between a point and a line, intercepts of a line and a circle, etc).	*line_line( ), line_arc( ), dist( ), compute_distance( ), compute_intercept( ), crossing( )*
get_encoder.c	Low-pass filters encoder value.	*get_encoder( )*
goals.c	Generate goal points from the data passed by the vision processor.	*transfer_to_world( ), cart_to_screen( ), left_side_angle( ), right_side_angle( ), generate_goals( ), good_vision_data( )*
golf.c	Generates inter-task communication message queues.	*golf( )*

init_disc.c	Handles initialization for the VMIO-14 discrete outputs.	*initialization_table_type( ), init_discretes( ), set_output, show_input( ), set_raw_pulse_width( ), set_steer_raw_pulse_width( ), forward_on( ), forward_off( ), reverse_on( ), reverse_off( )*
initial.c	Handles initialization when ALX first starts up.	*initialization( )*

Table D-1 Program List (continued)

Program	**Function**	**Module Included**
interface.c	Controls steering, brake and main drive motor. Also provides drivers to the sonar sensors.	*interface( ), send_ticks( ), switch_direction( ), switch_steering_direction( ), switch_brake( ), speed( ), steer( ), brake( )*
map_op.c	Handles operation conducted on the virtual map.	*initialize_map, update_pixel( ), pzero( ), realign( )*
receive.c	Receive sonar data and platform data messages, and updates the virtual map.	*receive ( )*
run.c	This is the main control program for ALX	*getstacks( ), freestacks( ), collision_imminent( ), plan_step( ), moveto( ), run( )*
search.c	Conducts alternative path search on the virtual map.	*measure( ), poisoned( ), tree( ), search( )*
sonar_cone.c	Registers a sensor scan on the virtual map.	*cone( ), computed_sector( ), mbr_y( ), x_intercepts( )*
sonar_op.c	Handles ultrasonic range sensor operation, include programming sensor firing sequence.	*init_sonar( ), changeshow( ), program_sonar( ), stop_sonar( ), start_sonar( ), sonar_range( )*
sonar_position.c	Provides information of ultrasonic range sensors' position and orientation on ALX.	*sonar_position( )*

stack_op.c	Handles stack operation.	*create_stack( ), stack_full( ), stack_empty( ), push( ), pop( ), peek( )*
vision_data.c	Conducts data communication between Datacube image processor and ALX MC68020 processor.	*ask_vision( ), receive_vision( ),*

Table D-2 Program List - Vision Algorithm

Program	Function	Module Included
BlobMain.c	Performs blob association and curve fitting. Calculates points on fit lines and passes data to the ALX control computer.	*AssociateBlobs(), ErrFcn(), FitLine(), TellGolf()*
IF_BlobData.c	Initializes and executes the Data Pipelines. Performs the thresholing selection and blob analysis.	*InitVision(), RunVision(), GetThreshold(), GetBlobs(), GetGoodBlobs()*

A set of header files were also created to support these C programs. The fundamental function of the header files is to provide definitions of various constants and new data types. These header file include: *VXWORKS.h, disc.h, map.h, z8536.h, av2receive.h, dr_lines.h, messages.h, cam_cal.h, golf.h, sonar_com.h, communication.h, initial.h,* and *sonar_op.h.*

These programs can be further divided into three categories in terms of their functionality. The three categories are as follows:

Category One: Programs that provide protocols for basic operation (e.g. A/D, D/A, mathematical calculation, etc). Programs like *math_func.c, geom.c, stack_op.c, adc.c, get_encoder.c* and*comm.c* belong to this category.

Category Two: Programs that conduct system initialization during system start-up process. These programs include: *golf.c, init_disc.c, initial.c, receive.c,* and *cali.c.*

Category Three: Programs that support ALX multi-tasking real-time control structure. These program together with the tasks they support are illustrated in Figure D-1
on next page.

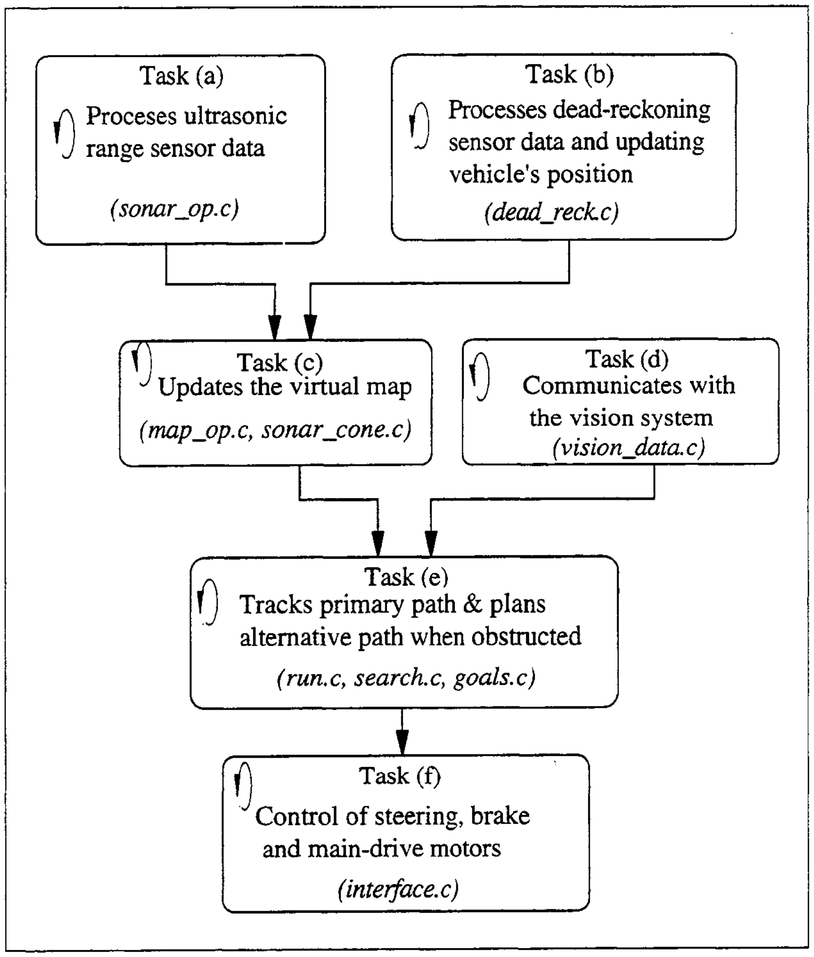

Task (a)
Proceses ultrasonic range sensor data
(sonar_op.c)
Task (b)
Processes dead-reckoning sensor data and updating vehicle's position
(dead_reck.c)
Task (c)
Updates the virtual map
(map_op.c, sonar_cone.c)
Task (d)
Communicates with the vision system
(vision_data.c)
Task (e)
Tracks primary path & plans alternative path when obstructed
(run.c, search.c, goals.c)
Task (f)
Control of steering, brake and main-drive motors
(interface.c)

Figure D-1 Programs that support multi-tasking control structure.