NPS52-86-025

NAVAL POSTGRADUATE SCHOOL

Monterey, California

A SIMULATION STUDY OF AN AUTONOMOUS STEERING SYSTEM FOR ON-ROAD OPERATION OF AUTOMOTIVE VEHICLES

Robert B. McGhee

Michael J. Zyda

Chiam Huat Tan

December 1986

Approved for public release; distribution unlimited

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by Contract from the United States Army Combat Developments Experimentation Center.

This report was prepared by:

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
NPS52-86-025

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
A SIMULATION STUDY OF AN AUTONOMOUS STEERING SYSTEM FOR ON-ROAD OPERATION OF AUTOMOTIVE VEHICLES

5. TYPE OF REPORT & PERIOD COVERED
Interim scientific

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Robert B. McGhee
Michael J. Zyda
Chiam Huat Tan

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Naval Postgraduate School
Monterey, CA 93943

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
MIPR No.
ATEC 88-86

11. CONTROLLING OFFICE NAME AND ADDRESS
HQ, USACDEC
Fort Ord, CA 93941

12. REPORT DATE
December 1986

13. NUMBER OF PAGES
182

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial intelligence
Robotics
Autonomous vehicles

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The study of human driving of automotive vehicles is an important aid to the development of viable autonomous vehicle navigation techniques. Observation of human behavior during driving suggests that this activity involves two distinct levels, the conscious and the unconscious.

Conscious actions relate to the logical behavior of a driver such as stopping the vehicle when a traffic light is red, slowing down the vehicle when it turns a bend, etc. Such behavior can be described using natural human language. The unconscious actions of a driver (cont'd on reverse) are much less obvious. There are many such activities occurring while we are driving a vehicle to a particular destination. One of the important unconscious efforts involves the selection of successive points on the road to steer the vehicle towards in order to achieve the desired road-following behavior. This research work attempts to mimic this unconscious behavior through the use of a computer simulation model.

This report represents the MS thesis work of the third author. Prof. McGhee served as thesis advisor and Prof. Zyda served as second reader during the conduct and documentation of this research.

A Simulation Study of an Autonomous Steering System for On-Road Operation of Automotive Vehicles

Chiam Huat Tan, Robert B. McGhee[1], and Michael J. Zyda

Naval Postgraduate School,
Code 52Mz, Dept. of Computer Science,
Monterey, California 93943

ABSTRACT

The study of human driving of automotive vehicles is an important aid to the development of viable autonomous vehicle navigation techniques. Observation of human behavior during driving suggests that this activity involves two distinct levels, the conscious and the unconscious.

Conscious actions relate to the logical behavior of a driver such as stopping the vehicle when a traffic light is red, slowing down the vehicle when it turns a bend, etc. Such behavior can be described using natural human language. The unconscious actions of a driver are much less obvious. There are many such activities occurring while we are driving a vehicle to a particular destination. One of the important unconscious efforts involves the selection of successive points on the road to steer the vehicle towards in order to achieve the desired road-following behavior. This research work attempts to mimic this unconscious behavior through the use of a computer simulation model.

This report represents the MS thesis work of the first author. Prof. McGhee served as thesis advisor and Prof. Zyda served as second reader during the conduct and documentation of this research.

[1] This author should be contacted if further information regarding this research is desired.

TABLE OF CONTENTS

I. INTRODUCTION

A. GENERAL BACKGROUND

Today there are many robots employed all over the world, especially in the USA and Japan, to do various kinds of industrial work. The benefits realizable through the use of these robots are numerous, easily attained, and, most importantly, proven. However, most of these industrial robots either lack any external sensory mechanisms or have only a few unsophisticated sensors [Refs. 1-3].

Robots that do not have any external sensors normally operate from a fixed position. They are programmed to perform a series of movements to accomplish a desired task. Once programmed, these robots repeat the programmed movements as many times as desired, accurately and precisely, regardless of the external environment.

The other kind of robot, the kind equipped with a few unsophisticated sensors, is able to perform limited sensing of the environment. Such a robot is capable of adapting to simple and small changes in the operating environment such as stopping when an unexpected object crosses into its path.

With the advent of the information age and microprocessor technologies, yet another kind of robot is becoming realizable. These kind of robots are called *autonomous* robots. Such robots are capable of making their own decisions and adapting quickly and safely "on-the-fly" to accomplish a mission [Refs. 4-5]. Such robots can either operate from a fixed position or from a mobile platform. In the case of mobile robots, the system can also be capable of avoiding obstacles along its navigation path. In this respect, a human being can be considered to be an extreme example of an ideal autonomous robot.

Humans receive information about their environment from two groups of sensors [Ref. 6]. One group provides conscious sensing and the other group provides subconscious sensing. The first sensing group comprises the five basic human sense organs: ears, nose, mouth, touch and eyes. Each of these sensors has a specialized processor attached to it which analyzes the sensor input. This is especially so with the eyes which provide humans with one of the most complex vision systems found in nature. A human is able to identify different objects under a wide variety of environmental conditions such as varying viewing distance, viewing angle, brightness, contrast, etc.

The second group of sensors provides *proprioceptive* information, which according to Thring [Ref. 7], gives an overall internal model of our body. It also provides *inertial* information from the vestibular system that senses body orientation, balance, and rotation.

All information received by the five basic sensors is sent to the highly complex cerebral cortex which acts as the central processor of the entire system [Ref. 7]. After processing the various sensory inputs, the cerebral cortex sends signals to all parts of the body including the cerebellum [Ref. 7].

The cerebellum is the chief coordinator of our motion system. Besides receiving information from the cerebral cortex about what movements are required, it also receives information from the vestibular system and the proprioceptors in order to be able to command muscular movements [Ref. 7].

The above oversimplified discussion of an ideal autonomous robot is intended to illustrate the complexities involved in building a real autonomous mobile robot. It must have a very elaborate and sophisticated sensory mechanism, numerous high-speed information processors working in parallel, and a very large and sophisticated controlling software system in order to achieve complete autonomy.

Finally, human beings achieve their mobility mainly through a pair of legs, but this is not necessarily so for an autonomous mobile robot. There are many ways for an autonomous mobile robot to achieve mobility on land. For local motion the alternatives are wheeled systems, tracked systems and legged systems [Ref. 8]. Of the three, the legged system is probably the most flexible but also the least understood method. There is a great deal of research currently underway on walking machines [Refs. 8-12].

Several attempts to build an autonomous wheel-based mobile robot were carried out as early as the 1960's [Ref. 13]. However, none of this research was able to deliver a completely autonomous robot. Some of these earlier works will be discussed in Chapter II. A number of research projects that are currently being carried out seem to be much more successful than their predecessors. The main reason for this could be that their predecessor's failures had prompted several related smaller research studies in areas such as sensor hardware, machine vision, and computer architectures to be carried out instead. The results of these research efforts are now being consolidated into the latest autonomous vehicle projects.

One of the areas which has received relatively little attention in previous and current research work on autonomous vehicles is that of human navigation and driving. This area may be pertinent to the viability of the future of autonomous vehicles, especially those on-road and wheeled-based. The objective of this work is to investigate and mimic a human driver driving a conventional automobile.

B. ORGANIZATION

Chapter II reviews some of the early research projects on autonomous mobile robots done in the late 1960's. Much of this work provides the necessary background for several autonomous vehicle research projects that are currently being carried out. Some of these later research projects are also summarized in this chapter.

The objective of this research work in relation to autonomous vehicles is discussed in greater depth in Chapter III. In this chapter, some of the basic aspects of conventional automotive vehicle mechanics are also explained. This is to show that the graphics simulation built for this study ignores many complex interactions that occur between a vehicle and its environment when the vehicle is moving. That is, a number of simplifying assumptions are made in this chapter so as to make the graphics simulation more feasible within the time constraints of this study. The mathematical model used for the graphics simulation is also derived and described in Chapter III.

The entire graphics simulation is implemented in C. The functions of the various modules developed for the simulation are described in Chapter IV. This chapter elaborates the overall software design strategy and the important issues that must be addressed when the software is to be improved or modified. The procedure for changing various vehicle gains is also described in this chapter.

Numerous experiments were conducted with the simulation model to support the validity of this work and the mathematical model used. Chapter V records and explains the results of all the experiments conducted using the simulation model.

The last chapter, Chapter VI, summarizes the work done and its benefit to autonomous vehicle research. Suggestions about some possible extensions to this research work are also given. This additional work would make the present study more comprehensive and substantiative.

Chapter VI is followed by a list of reference material used for this study. Finally, the graphics simulation source code is attached as an appendix.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

Research in autonomous systems began as early as the late 1960's [Ref. 13]. Many of these early research efforts did not fully materialize, mainly because of the technological limitations existing at that time. The outcome of these investigations, however, indicated that the complexity involved in certain areas such as image processing, scene analysis, planning, etc., required more work before further attempts to build autonomous systems could continue.

Since then, major advances and significant breakthroughs in several areas of technology have made many tasks which were difficult to implement or even not possible in the 1960's become more feasible. Improvements in VLSI technology allow very complex algorithms to be constructed in hardware. Miniaturization reduces the overall weight of vehicle controllers and also greatly increases electronic speed. New techniques in image processing and vision analysis enable very complex images to be examined. More details can now be extracted from images with these techniques than was previously possible [Refs. 14,15]. New computer architectures provide the massive computation power required for real-time processing [Refs. 16,17]. Large amounts of sensor data and images can now be reduced quickly to facts that are needed for autonomous decision making. New techniques developed in artificial intelligence provide more opportunities for autonomy. These techniques are capable of handling more complex knowledge representations and manipulations [Refs. 18,19].

B. AUTONOMOUS MOBILE ROBOTS

1. Shakey (1967)

The Shakey project represented one of the earliest attempts to study autonomous navigation using a mobile robot. Shakey was developed by Stanford Research Institute to study the real-time control of a robot system that interacts with a complex environment [Ref. 13].

Shakey moved around with two independently controlled wheels mounted on both sides of the vehicle. It had a rotatable "head" which carried a vidicon camera which provided "sight" to Shakey. This head was also provided with an optical range-finder. Several touch sensors were attached around the vehicle for collision detection and avoidance.

An SDS-940 computer was used to control the behavior of Shakey using two radio links. One link was used for telemetry and the other link was used for transmission of the visual input from the camera to the computer.

Another significant feature of this early attempt at an autonomous system was its man-machine interface. Commands could be given in primitive English that was analyzed and translated into the appropriate machine actions by a LISP program [Refs. 20,21]. This feature also included a simple question-answer capability.

Shakey's world consisted of a grid model and a property list model. The grid model is a hierarchically organized system of four-by-four grid cells. This model was constantly updated by the vision system and it served primarily as a free space map used for path planning and navigation.

The property list model kept track of the various characteristics of the objects in its world. Information about each object such as its coordinates, size, etc., gave Shakey a better sense of the world. Using this model, Shakey could navigate to a known object described in the property list. The model also provided information for collision-free navigation around the environment.

2. Stanford Cart (1973)

The research work for the Stanford Cart was carried out in the Stanford University Artificial Intelligence Laboratory [Ref. 22]. The only sensor installed on the cart was a camera remotely linked to a DEC KL-10 computer. The KL-10 computer functioned as the vehicle controller and also as an image processor. After each cart move, it received nine scene images from a slide-mounted camera, each taken from a different camera position. Distinctive features were extracted from the first image and this information was used with the rest of the images to perform a 3-D analysis of the scene in front of the cart.

The perceived scene was used by the navigation software to compute a collision-free path towards the goal. The collision-free path was determined by translating obstacles into circles on the floor and moving the cart, which is represented by a three meter circle, among these obstacle circles.

The Stanford Cart was able to maneuver successfully in a cluttered environment. However, it took a very long time to accomplish its mission [Ref. 22], typically requiring several hours to move a few tens of meters. This was largely due to the lengthy 3-D scene analysis processing time needed to determine the next cart move.

3. Hilare (1977)

This system was constructed in France at the Laboratoire d'Automatique et d'Analyse des Systemes in Toulouse. The purpose of the mobile robot was to serve as a testbed for general research in robotics and in robot perception and planning [Ref. 23].

The Hilare robot had a 3-D vision system consisting of a laser rangefinder and a video camera. A set of 14 ultrasonic emitters-receivers provided range data around the robot for distances of up to two meters, and a variety of other sensors provided other information required for autonomous navigation.

On-board 8085/86 microprocessors were used by Hilare to process sensor inputs. These processors were remotely connected to an SEL 32-77/80 computer which handled navigation and coordination. The SEL 32 was in turn remotely connected to another larger computer, an IBM 30/33, for higher level planning and control. A remote IBM-370 was also used for performing complex analysis tasks.

4. Robart I (1980)

This robot was built by LCDR Hobart R. Everett under the supervision of Professor R.E. Newton of the Naval Postgraduate School's Department of Mechanical Engineering. The aim of this project was to provide a development and demonstration platform for microprocessor-controlled mechanical systems [Ref. 24]. Patrolling was the main role of this robot. It was able to detect a variety of household dangers such as smoke, fire, toxic gas, flooding conditions, or intrusion, and was capable of then informing the user appropriately.

The Robart I system moved around on a tricycle wheelbase with front wheel control. It had a rotating "head" for scanning the environment. One of the major and important sensors missing from this robot is vision. Lacking vision, it had a forward-looking ultrasonic ranging unit, a long-range near-infrared proximity detector, ten short-range near-infrared proximity detectors, tactile feelers, and bumper switches. The last two groups of sensors were used for collision detection and avoidance.

To detect a person, the robot had a true-infrared (long wavelength infrared) body heat sensor. This sensor had a range of about fifty feet. The robot was also able to steer towards the center of a doorway with the help of a near-infrared long-range proximity sensor. It also had an assortment of other sensors for detecting flooding, fire, smoke and toxic gas conditions.

A surprising addition to this mobile robot was its speech capability with a two hundred and eighty word vocabulary. This allowed the robot to use voice communication to inform the user of any dangerous conditions. It also made use of this facility to report some of its internal status information.

The only computing machinery installed on Robart I was a SYM-1 microcomputer. This on-board computer used a 6502 microprocessor to which all the robots sensors were connected except for the speech synthesizer which had its own dedicated processor.

During normal operation, Robart I moved straight ahead and stopped at various points to perform surveillance. However, when an obstacle was detected, it would move to the left or right depending on which was appropriate and then continue its straight ahead movement.

5. Robart II (1982)

Robart II is an improved version of Robart I [Ref. 25]. The basic purpose of this new robot remains the same as for Robart I. However, not only are there more sensors on Robart II, but the number of different types of sensors installed also has been increased. This machine currently has six ultrasonic range-finders, fifty near-infrared proximity detectors, a long range near-infrared range-finder, and various other sensors used to detect smoke, fire, toxic gas, flooding conditions and intrusion.

The previous Robart used only one microcomputer for all of its processing requirements, which proved to be insufficient. Robart II has eight 65C02-based microcomputers to handle the increased number of sensors and also to provide more parallel processes to make it more responsive.

The eight microprocessors of Robart II are connected in a hierarchical fashion, each with a dedicated function to perform. The head, the drive motors, and the vision subsystem are each handled by a dedicated processor. The sonar and the speech subsystem are each handled by another hierarchy of two processors. All these processors, however, work under the direction of the SYM-1 computer.

C. AUTONOMOUS LAND VEHICLES

1. FMC Corporation Autonomous Vehicle (1985)

The vehicle used in this project is a 10-ton M113A2 armored personnel carrier, which is a tracked vehicle rather than a conventional wheeled vehicle. The vehicle carries an inertial navigation system. a vehicle controller computer, a sonic imaging sensor, and a master control computer. Beside this, a remotely located control trailer contains a Symbolics 3600 Lisp machine for the Planner software, a Sun workstation for the Mapmaker-Observer-Pilot, an IBM PC for sonic-sensor post-processing, and appropriate communications equipment [Ref. 26].

The FMC vehicle control software consists of five major interconnected subsystems that are called Planner, Observer, Mapmaker, Pilot, and Vehicle Control [Ref. 26]. Each of these subsystems has a well-defined and important function to perform. Together they make the FMC model one of the most advanced and successful autonomous land vehicles.

The FMC autonomous vehicle world is represented by digitized maps. These maps have terrain elevation and feature information that is used by the Planner. The primary role of the Planner is to generate segmented "freeways" defining free space from the starting position to the destination [Ref. 27]. The Planner is also capable of accepting mission requirements to decide the global path. Such requirements may include minimizing detection of the vehicle by the enemy. Another possibility is to minimize the time or energy involved to accomplish the mission.

The segmented freeway is used by the Observer. The Observer makes use of various sensors such as a sonic imaging sensor for obstacle detection and an inertial land-navigation system for calculating position and heading. With the input from the Planner and data from the sensors, the Observer derives a more usable plan for the next subsystem, the Mapmaker.

The Mapmaker generates the Pilot Map containing the vertices of a polygonal representation of the global path border, nearest obstacle borders, and sensor visibility limits. This is achieved by combining the Observer's input with the Obstacle Map. The latter is the product of the sonic imaging sensor.

The Pilot Map is very detailed but also very localized. It is used by the Pilot whose main responsibility is to guide the vehicle along an optimum path that is determined dynamically. To determine the optimum path, the Pilot generates several subpaths that are weighted according to certain criteria. Once the optimal path is picked, the Pilot issues the necessary instructions to the final subsystem, the Vehicle Controller, for actual execution.

A substantial amount of effort has been placed on the problem of obstacle avoidance in the FMC program. When an obstacle is encountered, the Pilot switches modes. It stops goal-seeking and starts obstacle-following. When the vehicle overcomes the obstacle, it resumes its goal-seeking mode.

2. Hughes Research Laboratory Autonomous Vehicle (1983)

The Hughes Research Laboratory autonomous vehicle is another state-of-the-art autonomous system. Like the FMC model, it too has a model of the world in which the autonomous vehicle is going to operate. Mission requirements and constraints are also accepted by the model to derive a plan for accomplishing the mission. Various sensors provide the required information about the environment for the vehicle to adapt dynamically to unforeseeable situations.

The entire system architecture of the Hughes system is based on a situation assessment module and an action planning module [Ref. 28]. The former uses available knowledge about the behavior and characteristics of a given object. Together with the interrelation between the various objects, it tries to visualize the surrounding environment. The latter module does the actual formulation of various actions required to fulfill the mission.

The most notable difference of this autonomous vehicle from other autonomous vehicle projects is the method of knowledge representation and the emphasis on applying artificial intelligence techniques. The system uses three types of stereotyped knowledge representations called *special problem solvers*, *scripts* and *domain-specific production rules* [Ref. 28].

The special problem solver consists of four path experts that decide which path to follow. The script based problem solver is used to provide predetermined ("canned") plans to solve problems that have stereotyped behavior. The rule-based system takes over whenever the script system cannot produce the proper actions to handle a situation.

The four path experts are called Shortest-path, Hide, Feasible-path, and Lost-path. Each is designed to cater to the requirements of an autonomous vehicle in various situations. The Shortest-path expert generates the shortest path between two points taking into account the obstacles between the two locations. The Hide expert determines a path with the best concealment characteristics. This path minimizes the vehicle's exposure to threats. The Feasible-path expert uses heuristics to produce a path between two locations. The heuristic procedure of this expert uses the information the vehicle has gathered along the path and the information about it's current environment to decide a feasible path for the vehicle to follow. Finally, the Lost-path module generates a path for the vehicle to explore and wander around the area when it has no path to the designated goal.

The hardware for the Hughes system includes a DEC 20 computer that implements a rule-based system for deciding the proper action for the vehicle to follow. Several Z80 computers are used to solve individual specialized problems such as finding the optimal path. Another processor generates commands to the vehicle control computer. Vision is handled by an image processing computer. A Lisp machine is used for for vehicle control. The vehicle carries a vidicon camera, ultrasonic sensors, touch sensors, and infrared ranging sensors.

3. Martin Marietta Autonomous Land Vehicle (1986)

The aim of this project is to demonstrate the state of the art in autonomous navigation and tactical decision making [Refs. 29,30]. The vehicle used, called the ALV (Autonomous Land Vehicle), is an eight-wheeled all-terrain vehicle from Standard Manufacturing, Inc. It is capable of traveling up to 18 mph on rough terrain and up to 45 mph on improved surfaces.

Mission goals and constraints specified to the ALV are interpreted by a Reasoning Subsystem. The output from the Reasoning Subsystem is a set of subgoals to be fulfilled in order to accomplish the mission. The Perception Subsystem controls all the sensors and generates a symbolic model of the environment for reasoning. The model consists of road boundaries. Moving the vehicle along the specified trajectory is handled by the Control Subsystem.

The Perception Subsystem sensors consist of an RCA color video CCD camera and a laser range scanner. The image taken by the camera is processed by a VICOM image processor. The output from this image is a set of 2-D edge points representing the two road boundaries. To generate a 3-D scene model for the Reasoning Subsystem, range information from the laser scanner is used by the image processor to compute the road boundaries in 3-D vehicle coordinates.

The Reasoning Subsystem consists of a goal seeker, a navigator, and a knowledge base. The goal seeker analyzes the mission given, and using the information in the knowledge base derives a sequence of activities for the vehicle to execute to achieve the goal. The navigator uses the 3-D model from the Perception Subsystem to compute several possible trajectories, and uses two cost functions to determine the trajectory for the vehicle to follow.

The pilot, which is part of the Control Subsystem, takes the specified trajectory the vehicle should follow and converts it into a sequence of steering commands to drive the vehicle.

The hardware architecture used in the ALV consists of an Intel multiprocessor system that has an 80286/80287 master processor, an 80816 navigation processor, an 8086 vehicle control processor, and an 8089 multichannel controller. The Perception Subsystem is managed by the VICOM image processor. However, this architecture will be affected by plans to use a more advanced computer, the BBN Butterfly parallel computer, to manage the increasing complexity of the Reasoning Subsystem. Another plan is to replace the currently heavily loaded VICOM image processor with a CMU WARP computer [Ref. 29].

D. SUMMARY AND CONCLUSIONS

In the control of autonomous land vehicles, the need for several high-performance and specialized processing systems working in parallel is fairly obvious. Without such elaborate hardware, software, and advanced computer architectures, an autonomous vehicle will not be able to fulfill even its most basic requirement, namely, to adapt quickly to environmental changes.

Many varieties of sensors, each capable of providing the vehicle with a means to sense the environment in a different manners are extremely important. An autonomous vehicle's ability to dynamically modify its behavior depends on the range and the capability of the various sensors installed in the vehicle. Without these sensors, autonomy would be very difficult to achieve if not impossible.

The effect of various gain settings in the vehicle controller could have a significant impact on the performance of an autonomous vehicle. With an improper gain, it was found in the FMC model that the vehicle fails to behave properly or performs poorly [Ref 26].

In order to avoid the problem of improper gain settings, a computer simulation allows the various vehicle controller gains to be adjusted easily. With this facility, different types of vehicle behavior can be simulated. The effect of these gain settings can be realistically visualized by means of the 3-D graphics simulation model.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

In this chapter, the model used for the 3D graphics simulation is described in detail. To assist a reader who has no control theory background, a brief description of the purpose of developing a mathematical model and its subsequent linearization is given.

The aim of this research is stated in the following section. This serves as a motivation for developing the graphics simulation. The reader should bear in mind that the hypothesis used in this research has not and may not be proven theoretically correct. The answer to this question requires much more work beyond what is presented in this work.

Many important interactions between the vehicle and the environment when the vehicle is moving have been neglected to keep the complexity of the mathematical model manageable. However, a short discussion of some of these interactions is included to give the reader a better appreciation of the real complexity involved.

The last section in this chapter provides a detailed derivation of the mathematical model used. All of the model linearization analysis is also presented.

B. AIM

The aim of this work is to examine and study by way of an "out of the vehicle windshield" graphics simulation model, a new technique for autonomous vehicle steering control. This technique attempts to mimic the way a human navigates his vehicle on the road.

The hypothesis is that a human driver unconsciously divides his route to his destination into "chunks" of small interconnecting line-of-sight segments. These segments are not prepared a priori. but instead are built dynamically while driving along a road. It is assumed that unconscious planning determines the distance of the next road segment from the current vehicle position. A point at the end of this road segment is then the driver's unconscious subgoal.

The location of a subgoal on the road depends on several factors such as the speed at which the vehicle is traveling, the road surface condition, the level of driving experience of the driver, and the general nature of the surrounding environment. The environment refers to situations such as the traffic conditions in front or behind the vehicle, the number of lanes available, and any potential danger spots ahead such as intersections, road bends, etc.

This work assumes that near-perfect vision is available for the autonomous vehicle and that the road is obstacle-free. These seemingly unreasonable assumptions were made to allow the author to concentrate on the aspect of unconscious driving rather than on the problems of image and vision processing, and obstacle avoidance, where there are currently numerous research activities being carried out by others [Refs. 14-19].

C. STATE SPACE REPRESENTATION

The aim of studying dynamic system behavior is generally one of gaining an understanding of the system, with a view to controlling it to satisfy a required specification [Ref 31]. A block diagram can be used to pictorially represent the system to be controlled. But to perform any analysis, a quantitative description is required and this may not be available from a block diagram. A system can be described quantitatively using a set of mathematical expressions which is commonly known as the mathematical model of the system. The two common methods to describe a system quantitatively are the *transfer function method* and the *state space method* [Ref. 31]. The latter technique is adopted in this work because it is more appropriate for computer simulation and it is also able to cope with more complex systems including nonlinear effects.

Most systems are inherently non-linear in nature. However, in many cases, a linearized analysis can be performed to predict the system behavior and to obtain suitable gain values for the actual model. Linearization basically involves restricting the values of the system variables to sufficiently small deviations from a datum point; i.e., the normal operating point of the system. A linearized system analysis of the vehicle steering problem is included in this chapter.

D. VEHICLE MECHANICS

1. Motivation

There are many complex interactions occurring between a moving vehicle and the environment that are ignored in this simulation. Some of these interactions are discussed here to provide the reader with some insight into the complexity involved.

2. Resistance to Motion

Vehicle acceleration arises when there is tractive effort between the tires and the road [Ref 32]. However, not all the tractive effort is used to provide acceleration; rather, a certain proportion of this effort is needed to overcome resistance to motion. The main sources of resistance to motion are air resistance, rolling resistance, and gradient resistance.

The amount of air resistance depends on numerous factors. The vehicle shape, size and its velocity are some of these factors. The interaction of the tires and the road surface give rise to *rolling resistance* which depends upon factors such as vehicle velocity, vehicle load, and the type of road surface. *Gradient resistance* arises only when the vehicle climbs a slope. The amount of gradient resistance is directly proportional to the steepness or gradient of the slope it is overcoming.

3. Slip Angle

When a wheel is rolling, it is acted upon by a side force due to imperfect contact between the tires and the road surface. The angular difference between the direction of motion and true wheel rolling direction is called the *slip angle* [Ref. 33]. When a vehicle is cornering, depending on the sign of the slip angle, the vehicle may *understeer* or *oversteer.*

4. Brakes

The amount of braking effort required is related to the load carried by the wheels and the coefficient of friction of the road surface. Another important consideration is the location of brakes. When brakes are applied while a vehicle is cornering, computation of the braking effort is more complex. This is due to the side forces acting on the wheels when the vehicle is cornering.

E. VISION MODEL

The vision model used in this graphics simulation is extremely simple. The model consists of a set of road points representing the center of the entire road circuit. When the vehicle is operating in the autopilot mode, it "sees" these points down the road. one of which is selected to be the new target point for the vehicle to steer towards.

F. SIMULATION MATHEMATICAL MODEL

1. A Simplified Planar Dynamic Model For Manual Control

The notation used in this work will follow as closely as possible to that adopted by Frank and McGhee [Ref. 34]. A top view of the vehicle is shown in Figure 3.1.

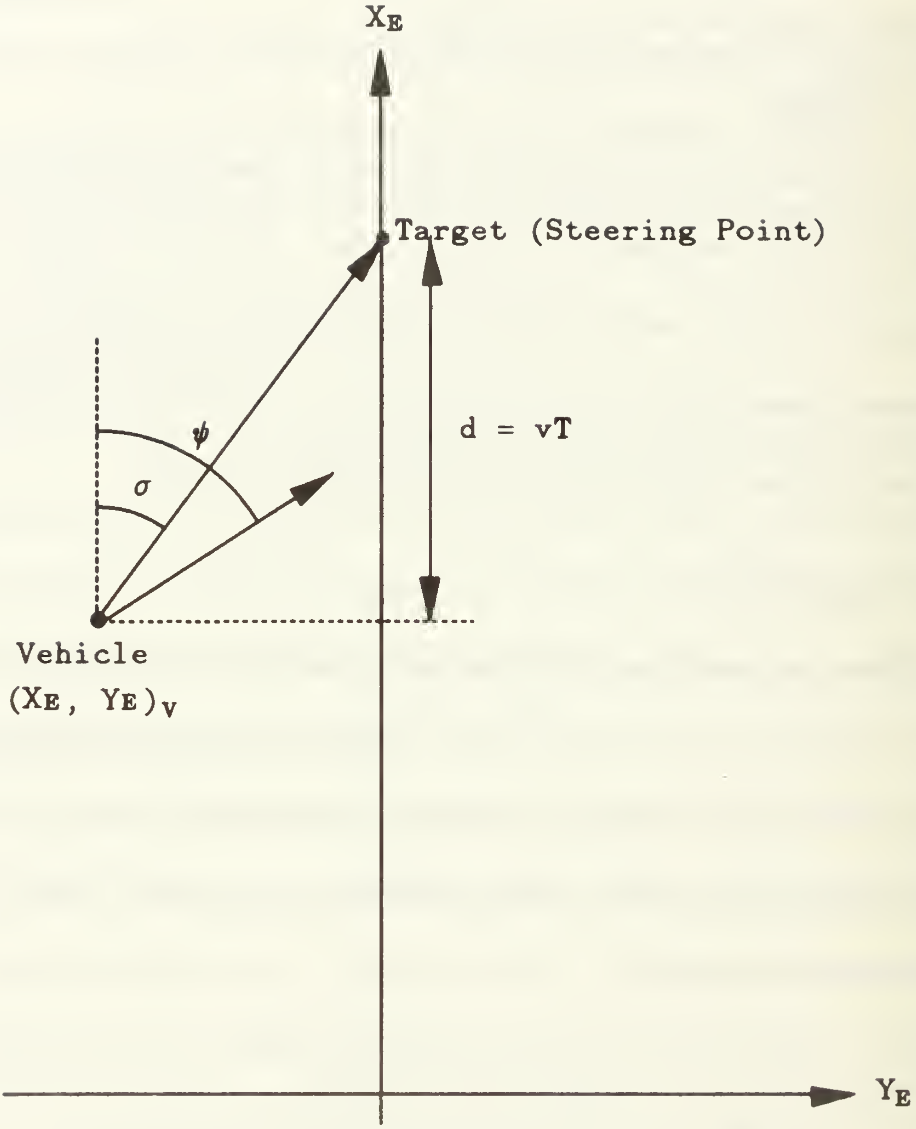


Figure 3.1 Top View Of The Vehicle

The vehicle will be confined to a flat road surface. Therefore z = 0, $\dot{z}$ = 0, and the position vector can be collapsed to a two-dimensional vector. The rotational moment of inertia is ignored. This means that the vehicle is idealized to a *point mass*. To further simplify the model, it is assumed that the velocity vector always lies along the vehicle x axis; i.e., no *sideslip* angle is allowed. Finally, it is assumed that the vehicle turning rate, $\dot{\psi}$, is linearly proportional to the forward velocity and to the steering wheel angle, θ. That is,

$$\dot{\psi} = k_{\dot{\psi}} \theta \dot{x} \tag{3.1}$$

To calculate the associated turning radius, R, note that the time to rotate the vehicle through an angle 2π is

$$t_{2\pi} = \frac{2\pi}{|\dot{\psi}|} = \frac{2\pi}{k_{\dot{\psi}}|\theta \dot{x}|} \tag{3.2}$$

while the distance traveled in this time is

$$d = 2\pi R = |\dot{x}|\, t_{2\pi} \tag{3.3}$$

Thus

$$R = \frac{|\dot{x}|}{2\pi} t_{2\pi} = \frac{|\dot{x}|}{k_{\dot{\psi}}|\theta \dot{x}|} = \frac{1}{k_{\dot{\psi}}|\theta|} \tag{3.4}$$

Or

$$k_{\dot{\psi}} = \frac{1}{R|\theta|} \tag{3.5}$$

This equation shows that large values for $k_{\dot{\psi}}$ correspond to "stiff" steering while small values correspond to "sloppy" steering.

Longitudinal control modeling accelerator control [Refs. 35,36] can be approximated by

$$\ddot{x} = \frac{1}{\tau_a}\left(\dot{x}_c - \dot{x}\right) \tag{3.6}$$

where $\dot{x}_c$ is the command velocity, which in turn is a function of the accelerator depression angle, and τ_a is the acceleration time constant. It is easily shown that for a step change in $\dot{x}_c$ at t = t_0 , the resulting velocity profile is

$$\dot{x}(t) = \dot{x}(t_0) + (\dot{x}_c(t) - \dot{x}(t_0))e^{-\frac{t-t_0}{\tau_a}} \tag{3.7}$$

Combining all of the above results, a suitable state vector for this system is

$$\vec{x} = (x_E,\, y_E.\, \dot{x},\, \psi)^T \tag{3.8}$$

If the *control vector* , provided by the human operator is defined as

$$\vec{u} = (\dot{x}_c,\, \theta)^T \tag{3.9}$$

then, from the above analysis, the component form of the state equation is:

$$\dot{x}(1) = \dot{x}_E = \dot{x}\cos\psi = x(3)\cos x(4) \tag{3.10}$$

$$\dot{x}(2) = \dot{y}_E = \dot{x}\sin\psi = x(3)\sin x(4) \tag{3.11}$$

$$\dot{x}(3) = \ddot{x} = -\frac{1}{\tau_a}x(3) + \frac{1}{\tau_a}u(1) \tag{3.12}$$

$$\dot{x}(4) = \dot{\psi} = k_{\dot{\psi}}\, x(3)\, u(2) \tag{3.13}$$

For manual control, $\vec{u}(t)$ is provided by the human operator. Eq. (3.10 - 3.13) can then be numerically integrated and provided to the operator in the form of a graphics display to permit him to guide the vehicle around a prescribed course. Note that for practical vehicles, both $\dot{x}_c$ and θ must have upper and lower bounds.

2. Pursuit Navigation and Small Angle Linearization Analysis

From the previous analysis, we have

$$\vec{x} = (x_E,\, y_E,\, \dot{x},\, \psi)^T \tag{3.15}$$

$$\dot{\vec{x}} = (\dot{x}_E,\, \dot{y}_E,\, \ddot{x},\, \dot{\psi})^T \tag{3.16}$$

For autopilot control, in the research of this study. the vehicle forward velocity is assumed to be constant. Therefore

$$\dot{x}(3) = 0 \tag{3.17}$$

One approach to steering the vehicle is to simply aim it directly at the current steering point. That is, the vehicle heading could in principle be governed by the simple relationship

$$\psi(t) = \sigma(t) \tag{3.18}$$

Such a steering law is sometimes called *pure pursuit navigation.* Obviously, if pure pursuit navigation could be realized, the vehicle would pass directly through each steering point. However, this does not fit the model of this chapter in which the steering point is at a constant distance d ahead of the vehicle. A potential solution to this problem is to differentiate Eq. (3.18) to obtain

$$\dot{\psi}(t) = \dot{\sigma}(t) \tag{3.19}$$

Unfortunately, referring to Figure 3.1, it can be seen that whenever $\psi = 0$, it will also be true that $\dot{\sigma} = 0$. In this case, the vehicle will simply move parallel to the road center and will not turn toward it. To remedy this problem, an *integral* term can be added to Eq. (3.19) resulting in the steering equation

$$\dot{\psi} = \dot{\sigma} + k_\sigma (\sigma - \psi) \qquad , k_\sigma > 0 \tag{3.20}$$

Referring to Figure 3.1, the "line-of-sight" angle, σ, is given mathematically by

$$\sigma = \tan^{-1}\left(\frac{Y_T - Y_E}{X_T - X_E} \right) \tag{3.21}$$

The corresponding line-of-sight rate, $\dot{\sigma}$, can be approximated by

$$\dot{\sigma}(t_j) = \frac{\sigma(t_j) - \sigma(t_{j-1})}{t_j - t_{j-1}} \tag{3.22}$$

where j is an index associated with successive computation cycles. Eq. (3.20) will be used in the following linearized system analysis.

Referring again to Figure 3.1, for the straight road case, and using small angle approximations, we have the following:

$$d \gg y_E \tag{3.23}$$

$$\sigma = -\frac{y_E}{d} \tag{3.24}$$

$$\dot{\sigma} = -\frac{\dot{y}_E}{d} \tag{3.25}$$

$$\psi = \frac{\dot{y}_E}{\dot{x}(0)} \tag{3.26}$$

$$\dot{\psi} = \frac{\ddot{y}_E}{\dot{x}(0)} \tag{3.27}$$

Using Eq. (3.24) - (3.27), the vehicle guidance law can be written as

$$\dot{\psi} = -\frac{\dot{y}_E}{d} + k_\sigma \left(-\frac{y_E}{d} - \frac{\dot{y}_E}{\dot{x}(0)} \right) \tag{3.28}$$

From Eq. (3.13)

$$\dot{\psi} = k_{\dot{\psi}}\, x(3)\, u(2) \tag{3.29}$$

Therefore the vehicle guidance law can also be written as

$$u(2) = \frac{\dot{\psi}}{k_{\dot{\psi}}\, x(3)} = -\frac{1}{k_{\dot{\psi}}\, x(3)} \left[\left(\frac{1}{d} + \frac{k_\sigma}{\dot{x}(0)} \right) \dot{y}_E + \frac{k_\sigma}{d} y_E \right] \tag{3.30}$$

From Eq. (3.27) and (3.28), the *linearized vehicle control equation* is

$$\frac{\ddot{y}_E}{\dot{x}} = -\frac{\dot{y}_E}{d} + k_\sigma \left(-\frac{y_E}{d} - \frac{\dot{y}_E}{\dot{x}} \right) \tag{3.31}$$

or

$$\ddot{y}_E = -\frac{\dot{x}}{d}\dot{y}_E - k_\sigma\, y_E \frac{\dot{x}}{d} - k_\sigma\, \dot{y}_E \tag{3.32}$$

As stated previously, for the purpose of linearized system analysis, it is assumed that the vehicle steers toward a point located in front of the vehicle at a constant distance d where

$$d = \dot{x}\; T \tag{3.33}$$

The quantity T is called the *steering point prediction time* and is evidently given by

$$T = \frac{d}{\dot{x}} \tag{3.34}$$

Using Eq. (3.33), Eq. (3.32) can be re-written as

$$\ddot{y}_E = -\left(k_\sigma + \frac{1}{T} \right) \dot{y}_E - \frac{k_\sigma}{T} y_E \tag{3.35}$$

or

$$\ddot{y}_E + \left(k_\sigma + \frac{1}{T} \right) \dot{y}_E + \frac{k_\sigma}{T} y_E = 0 \tag{3.36}$$

The characteristic equation [Ref. 37] associated with Eq. (3.36) is

$$\lambda^2 + k_1 \lambda + k_0 = 0 \tag{3.37}$$

where

$$k_1 = \left(k_\sigma + \frac{1}{T} \right) \tag{3.38}$$

and

$$k_0 = \frac{k_\sigma}{T} \tag{3.39}$$

The corresponding response of the system to initial condition errors is, for the case of distinct eigenvalues,

$$y(t) = c_1 e^{\lambda_1 t} + c_2 e^{\lambda_2 t} \tag{3.40}$$

where λ_1 and λ_2 are the roots of Eq. (3.37), c_1 and c_2 are determined from the boundary conditions $y(t_0)$, $\dot{y}(t_0)$, and t_0 is the time when autopilot is turned on.

Critical damping [Ref. 38] results when the eigenvalues λ_1 and λ_2 are equal, real, and negative. Critical damping implies the most rapid response possible to steering errors without overshooting the road center line in response to an initial position error. From Eq. (3.37), the system eigenvalues are

$$\lambda = -\frac{k_1}{2} \pm \left[\left(\frac{k_1}{2} \right)^2 - k_0 \right]^{\frac{1}{2}} \tag{3.41}$$

Since critical damping results when the second term in this equation is zero, it follows that

$$k_0 = \frac{{k_1}^2}{4} \tag{3.42}$$

Substituting Eq. (3.38) and Eq. (3.39) into Eq. (3.42),

$$\frac{k_\sigma}{T} = \frac{1}{4}\left(k_\sigma + \frac{1}{T}\right)^2 \tag{3.43}$$

Solving this equation yields the following relationship:

$$k_\sigma T = 1 \tag{3.44}$$

Using this relationship in Eq. (3.41),

$$\lambda = -k_\sigma \tag{3.45}$$

From this, the system *total time constant*, τ_{total}, [Ref. 38] is given by

$$\tau_{total} = \left(\frac{1}{\lambda_1} + \frac{1}{\lambda_2}\right) = \frac{2}{k_\sigma} \tag{3.46}$$

which means that vehicle position error will be corrected to about 40% of its initial value in approximately τ_{total} seconds [Ref. 39]. As a specific example, if $T = 1$ is chosen. then $k_\sigma = 1$, $\lambda = -1$, and $\tau_{total} = 2$. Another example, if $T = 2$ is chosen, then $k_\sigma = 0.5$, $\lambda = -0.5$, and $\tau_{total} = 4$. This completes the derivation of parameter values for this example of a vehicle steering equation.

3. Proportional Navigation With Integral Term

In Chapter V, it is shown that while pursuit navigation performs as predicted by the above linearization for driving on a straight road, it tends to steer to the inside of turns on a curved roadway. One way to solve this problem is to introduce an additional gain term, $k_{\dot{\sigma}}$, which multiplies $\dot{\sigma}$. This result is a form of *proportional navigation* [Ref. 40] with the addition of the integral term introduced in the previous section of this chapter.

In order to determine a value for $k_{\dot{\sigma}}$ suitable for driving on a curved road, it should be noted that the resulting vehicle guidance law is:

$$\dot{\psi} = k_{\dot{\sigma}}\dot{\sigma} + k_{\sigma}(\sigma-\psi) \tag{3.47}$$

and that the condition for steady turn is $\dot{\psi} = \dot{\sigma}$. Thus, in such a situation,

$$\dot{\sigma} = \dot{\psi} = k_{\dot{\sigma}}\dot{\sigma} + k_{\sigma}(\sigma-\psi) \tag{3.48}$$

so

$$\sigma-\psi = \frac{1-k_{\dot{\sigma}}}{k_{\sigma}}\dot{\sigma} \tag{3.49}$$

From Figure 3.1, the road/vehicle equation for a curved road is

$$x_E{}^2 + (y_E - R)^2 = R^2 \tag{3.50}$$

or

$$x_E{}^2 + y_E{}^2 - 2y_E R = 0 \tag{3.51}$$

which can be approximated by

$$x_E{}^2 = 2y_E R \tag{3.52}$$

Using small angle approximations,

$$x_E = \dot{x}T \tag{3.53}$$

and thus, on a curved road with radius R,

$$y_E = \frac{x_E{}^2}{2R} = \frac{\dot{x}^2 T^2}{2R} \tag{3.54}$$

The value for σ is thus approximately

$$\sigma = \frac{y_E}{x_E} = \frac{x_E}{2R} = \frac{\dot{x}T}{2R} \tag{3.55}$$

and to stay on the road,

$$\dot{\psi} = \frac{\dot{x}}{R} \tag{3.56}$$

Since, in a steady turn, $\dot{\sigma} = \dot{\psi}$, combining Eq. (3.49), (3.55), and (3.56) it follows that

$$\sigma = \frac{\dot{x}T}{2R} = \frac{1-k_{\dot{\sigma}}}{k_\sigma}\frac{\dot{x}}{R} \tag{3.57}$$

or

$$T = 2\,\frac{1-k_{\dot{\sigma}}}{k_\sigma} \tag{3.58}$$

Eq. (3.58) is one of the constraints for this model.

4. Linearized Analysis For Proportional Navigation

Using small angle approximations, with the inclusion of $k_{\dot{\sigma}}$ and the road curvature, Eq. (3.36) becomes

$$\frac{\ddot{y}_E}{\dot{x}} = k_{\dot{\sigma}} \frac{\dot{y}_T - \dot{y}_E}{\dot{x}T} + k_\sigma \left[\frac{y_T - y_E}{\dot{x}T} - \frac{\dot{y}_E}{\dot{x}} \right] \tag{3.59}$$

or

$$\ddot{y}_E + \left[\frac{k_{\dot{\sigma}}}{T} + k_\sigma \right] \dot{y}_E + \frac{k_\sigma}{T} y_E = \frac{k_{\dot{\sigma}}}{T} \dot{y}_T + \frac{k_\sigma}{T} y_T \tag{3.60}$$

The characteristic equation associated with this result is

$$\lambda^2 + k_1 \lambda + k_0 = 0 \tag{3.61}$$

where

$$k_1 = \frac{k_{\dot{\sigma}}}{T} + k_\sigma \tag{3.62}$$

$$k_0 = \frac{k_\sigma}{T} \tag{3.63}$$

Substituting constraint Eq. (3.58), it follows that

$$k_1 = \frac{k_{\dot{\sigma}} k_\sigma}{2(1 - k_{\dot{\sigma}})} + k_\sigma \tag{3.64}$$

$$= k_\sigma \left(\frac{2(1 - k_{\dot{\sigma}}) + k_{\dot{\sigma}}}{2(1 - k_{\dot{\sigma}})} \right) \tag{3.65}$$

$$= k_\sigma \left(\frac{2 - k_{\dot\sigma}}{2 - 2k_{\dot\sigma}} \right) \tag{3.66}$$

and

$$k_0 = \frac{k_\sigma^2}{2 - 2k_{\dot\sigma}} \tag{3.67}$$

Referring to Eq. (3.61)

$$\lambda = -\frac{k_1}{2} \pm \left[\left(\frac{k_1}{2} \right)^2 - k_0 \right]^{\frac{1}{2}} \tag{3.68}$$

$$= -k_\sigma \frac{2 - k_{\dot\sigma}}{4 - 4k_{\dot\sigma}} \pm k_\sigma \left[\left(\frac{2 - k_{\dot\sigma}}{4 - 4k_{\dot\sigma}} \right)^2 - \frac{2}{(4 - 4k_{\dot\sigma})} \right]^{\frac{1}{2}} \tag{3.69}$$

$$= -k_\sigma \left[\frac{2 - k_{\dot\sigma}}{4 - 4k_{\dot\sigma}} \pm \frac{1}{4-4k_{\dot\sigma}} \left(4 - 4k_{\dot\sigma} + k_{\dot\sigma}^2 - 8 + 8k_{\dot\sigma} \right)^{\frac{1}{2}} \right] \tag{3.70}$$

$$= -\frac{k_\sigma}{4 - 4k_{\dot\sigma}} \left[(2 - k_{\dot\sigma}) \pm \left(k_{\dot\sigma}^2 + 4k_{\dot\sigma} - 4 \right)^{\frac{1}{2}} \right] \tag{3.71}$$

Thus, for the critical damping condition,

$$k_{\dot\sigma}^2 + 4k_{\dot\sigma} - 4 = 0 \tag{3.72}$$

or

$$k_{\dot{\sigma}} = -2 \pm \left(2^2 + 4\right)^{\frac{1}{2}} \tag{3.73}$$

Solving Eq. (3.73) with the constraint $k_{\dot{\sigma}} > 0$,

$$k_{\dot{\sigma}} = 0.828 \tag{3.74}$$

From Eq. (3.71)

$$\lambda = -\frac{k_\sigma \left(2 - k_{\dot{\sigma}}\right)}{4 - 4k_{\dot{\sigma}}} \tag{3.75}$$

so from Eq. (3.74)

$$\lambda = -k_\sigma \frac{1.172}{0.688} = -1.703\ k_\sigma \tag{3.76}$$

and from Eq. (3.58)

$$T = \frac{0.344}{k_\sigma} \approx \frac{1}{3k_\sigma} \tag{3.77}$$

or

$$k_\sigma T \approx \frac{1}{3} \tag{3.78}$$

As an example of the application of the above results, if $\lambda = -0.5$ is chosen, then $k_\sigma = 0.294$, $T = 1.17$, $k_{\dot{\sigma}} = 0.828$. and $\tau_{total} = 4$. For another example, if $T = 0.8$ is chosen, then $k_\sigma = 0.43$, $\lambda = -0.73$, $k_{\dot{\sigma}} = 0.828$. and $\tau_{total} = 2.7$.

G. SUMMARY

The aim of this study is established. The assumptions made have been delineated so that a mathematical model can be developed for the realization of a computer simulation to study both manual and automatic steering of a highway vehicle.

In the next chapter, the main concern is the actual implementation of the 3D graphics simulation using the mathematical model derived in this chapter.

IV. COMPUTER SIMULATION MODEL

A. INTRODUCTION

Various methods of implementing the mathematical model developed in the previous chapter were examined during the formulation of this study. It was finally decided that the best way to perform the simulation would be to have a 3D color graphics animation model which can be driven by the user. One of the most suitable machines available at the Naval Postgraduate School for this purpose is the IRIS (Integrated Raster Imaging System) color graphics system. A brief description of this graphics system configuration and its hardware features is given in this chapter.

In what follows, much attention is devoted to the man-machine interface of the simulation. The user can drive the simulation with a mouse attached to the graphics system. He can also use the keyboard to turn on or off certain information displays concerning the status of the simulation.

A complete description of all the modules and supporting files which are used for the simulation is provided in this chapter for those who plan to study the simulation in detail. A user guide is also included, though it is not absolutely necessary to read it wholly in order to run the simulation.

B. IRIS-2400 WORKSTATION

1. Hardware And Overall System Description

The IRIS graphics workstation installed in the Naval Postgraduate School Graphics Laboratory is a high-performance, high-resolution 1024 x 768 color graphics system. A combination of custom VLSI circuits, conventional hardware, firmware, and software provide a very powerful set of graphics commands to perform 2D and 3D graphics [Refs. 41-44]. There are currently two IRIS systems installed in the laboratory. Both systems are Unix-based machine but one system has a Motorola MC68010 processor with 5MB of CPU memory while the other system has a Motorola MC68020 processor with 6MB of CPU memory. The configuration of both IRIS-2400 workstations consist of an electronic cabinet with two 72 MB Winchester disk drives, an 83-key up-down encoded keyboard, a three-button mouse, a high-resolution 60 Hz non-interlaced 19-inch RGB color monitor, 32 bitplanes, a hardware matrix multiplier Geometry Pipeline, and a floating point accelerator.

The IRIS hardware consists of three pipelined components. It is this design structure that makes the IRIS different from many other graphics systems. Many systems tend to implement their graphics capabilities with software that is cheaper. However, these systems have much lower efficiency and much lower performance. Also with these systems, the 3D color graphics simulation model would require significantly more programming effort and time, not considering the fact that the final overall performance may not be suitable for this simulation.

The three pipelined components in the IRIS system are the applications/graphics processor, the Geometry Pipeline, and the raster subsystem [Ref. 41].

Graphics commands are processed by the applications/graphics processor. Commands are first sent through the Geometry Pipeline, which performs matrix transformations on the coordinates, clips the coordinates to normalized coordinates, and scales the transformed, clipped coordinates to screen coordinates. The raster subsystem accepts the output of the Geometry Pipeline. It fills in the pixels between the endpoints of the lines. fills in the interiors of polygons, converts character codes into bit-mapped characters. and performs shading, depth-cueing, and hidden surface removal. The system maintains a color value for each pixel in its bitplanes which determines the image color on the monitor. A total of thirty-two bitplanes allow color graphics images to be presented in a very realistic way.

2. Programming Language

The IRIS system software is written in C, but the commands in the graphics Library are callable in C, FORTRAN, Pascal, and Extended Common Lisp (ExCL). However, at the time when the simulation was implemented, only Pascal and C were available. Consequently, C was chosen to be the programming language for implementing the simulation. One of the reasons for this decision was the programming experience of the author with C and the other was to maintain compatibility with the IRIS system software. However, C is not without its disadvantages. One of problems with C is that it is a *weakly typed* language. This tends to make software development more frustrating and time consuming.

This frustration could have been considerably alleviated if a ***strongly typed*** language like Pascal had been used.

3. Graphical Objects

This is a group of drawing commands used to defined a geometric model or an object. The advantage of using these commands is that the graphical objects can then be treated as a single entity which can be moved, scaled, rotated, or combined with other graphic objects to form more complex objects.

4. Double Buffering

The screen image in an IRIS system is stored in a set of bitplanes. Each bitplane provides one bit of storage per pixel. An RGB value is associated with each pixel which determines the color and the brightness of the pixel. This value is made up of three eight-bit intensity values - one for red, one for green, and one for blue.

The bitplanes can be used in either of the two modes, single buffer or double buffer. In the single buffer mode, up to twelve bitplanes can be used to handle the image color and the rest can be used for *z-buffering* , a technique used to remove hidden lines and surfaces. The problem with single buffering is that the image on the screen is simultaneously updated and displayed. This means that incomplete or changing picture may appear on the screen.

In double buffering mode, the bitplanes are divided into two portions, called *front* and *back buffers.* The purpose of having two buffers is to have one buffer being updated while another buffer is being displayed. The benefit of this arrangement is that a changing or incomplete image will not appear on the screen. This is important in certain applications such as motion animation. The 3D graphics simulation uses the bitplanes in double buffering mode.

5. Coordinate Transformation

In order to manipulate graphical objects, coordinate transformations are required [Ref. 41]. When defining the object, it is convenient to chose a point to be the object origin and to then build the object around this selected reference point. The space which the object occupies is called *object space.*

The object space must be transformed into *world space* when a group of objects is to be displayed together. Since the world space can be viewed from various directions and orientations, another coordinate system called *eye space* is required to specify how the world space is to be viewed. Finally, this eye space must be mapped into *screen space* which is a 2-D coordinate system for displaying the objects on the graphics screen.

Four types of transformation commands are available on the IRIS to perform the various mappings described above:

o *Modeling transformation* commands, such as rotate, translate, and scale, transform the coordinate system of objects.

o *Viewing transformation* commands, such as polarview and lookat, place the viewer and eye coordinate system in world space.

o *Projection transformation* commands, such as perspective, window, ortho, and ortho2, transform eye space to the screen coordinate system.

o *Viewport transformation* commands, such as viewport and scrmask, define the position of the rectangular region on the screen to be used for displaying the image.

C. USER GUIDE

To run the graphics simulation, just enter the following command:

carsimu

It takes a short time for the computer to read the roadmap into the memory.

The simulation begins with the display as shown in Figure 4.1. As can be seen, the top half of the graphics display is an "out-of-the-windshield" view of the world and the lower half of the display is the vehicle dashboard display area.

On the extreme left of the dashboard display is some information about how to drive the vehicle. This display area shows that pressing the three mouse buttons simultaneously terminates the simulation.

Pressing the right mouse button is equivalent to stepping on the accelerator of a conventional vehicle except that every mouse click increases the desired speed by a fixed increment of four kilometers per hour. Pressing the middle mouse button is similar to stepping on the brake of a conventioral vehicle except that every mouse click decreases the desired speed by a fixed decrement of four kilometers per hour. Pressing the left mouse button, which is the third and last button, stops the vehicle.

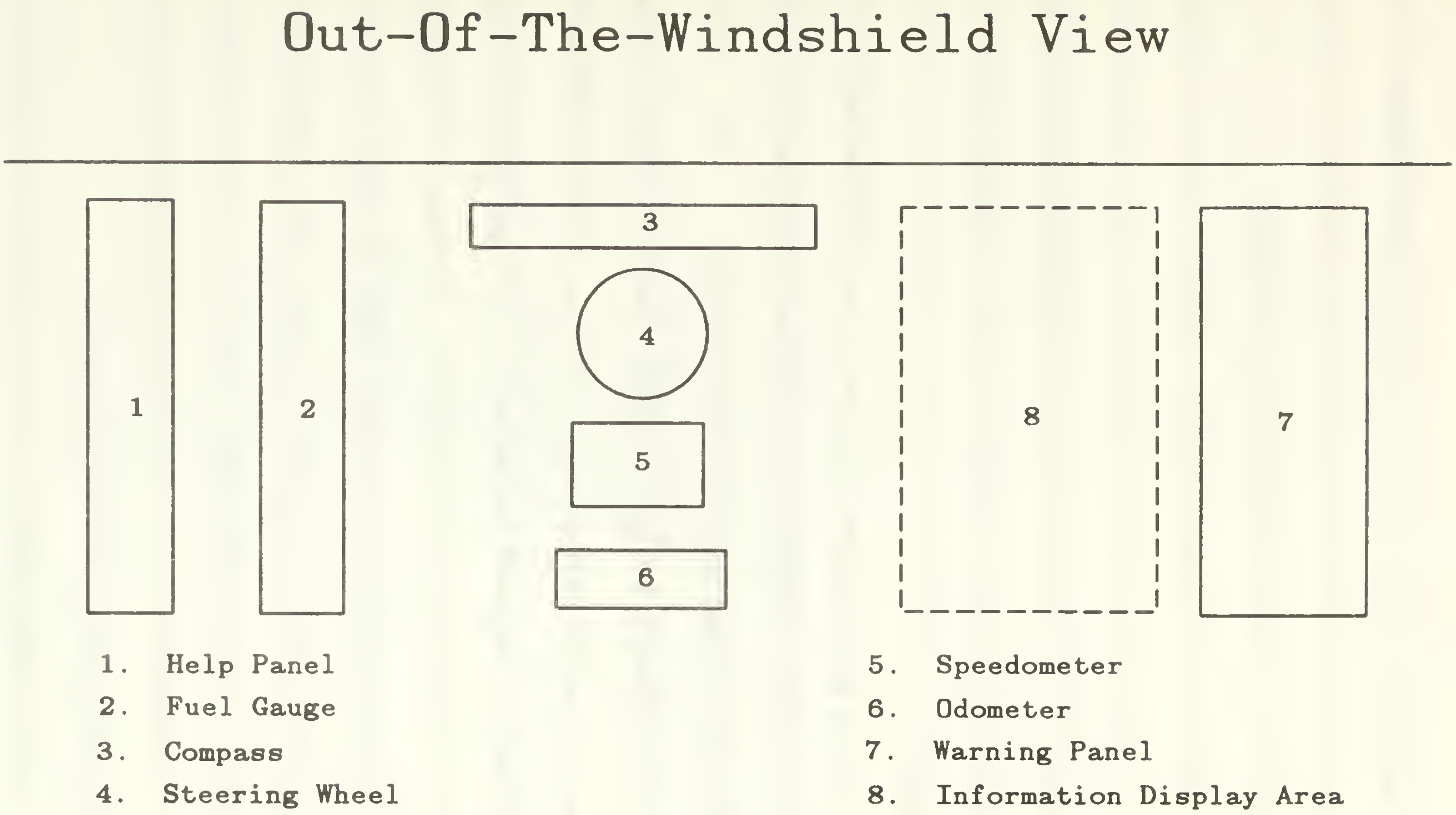


Figure 4.1 Graphics Simulation Display

Moving the mouse to the left or to the right corresponds to turning the steering wheel of a vehicle. The steering wheel turning rate is controlled by the speed the mouse is moved towards the left or the right.

Besides using the mouse buttons for driving the vehicle, the keyboard keys are used to toggle various information displays or to reset certain dashboard displays. Pressing h or H on the keyboard stops the simulation temporarily and the whole dashboard area is used to display additional information about the various key functions.

The fuel gauge indicates the amount of fuel left in the vehicle. When the fuel runs out before the whole circuit is completed, the simulation stops and a message is displayed to inform the driver of the condition.

The compass is located on the top center of the dashboard display. This shows the vehicle heading angle. Following this is the steering wheel display indicating the position of the steering wheel, the speedometer showing the current velocity of the vehicle, and the odometer recording the total distance traveled. The first three indicators are especially helpful for manual driving since it allows the driver to "feel" the situation and make more appropriate corrections if necessary. In real driving, these indicators are not as important because the driver's kinesthetic senses provide him with information concerning the steering wheel and the various forces acting upon him.

On the extreme right of the dashboard display is the warning panel. When the vehicle is not moving, the brake light is turned on. When the engine is warmed up, the temperature light shows yellow and when it overheats, the light turns red. The most important part of the warning panel is the danger light. This area blinks when the vehicle moves too close to the edge of the road. An alarm will also be given if it is switched on by the user with one of the keys on the keyboard.

The last area on the dashboard display is called the information display area. The main purpose of this area is to show some key technical data used for the current simulation run. This area, by default, is turned off and it can be turned on with a key on the keyboard.

A clock indicating the date and time of day is displayed on the top left of the graphics display. Again this can be turned off with a key. When the vehicle is operating in the autopilot mode, a blinking indicator is displayed on the top center of the graphics display.

D. MODULE DESCRIPTION

1. Carsimu.c

This is the main module of the entire graphics simulation. It sets up the system by initializing all the local and global variables and the graphics facilities. After setting up, the actual simulation is controlled from this module. Figure 4.2-4.8 show the flowcharts of this module.

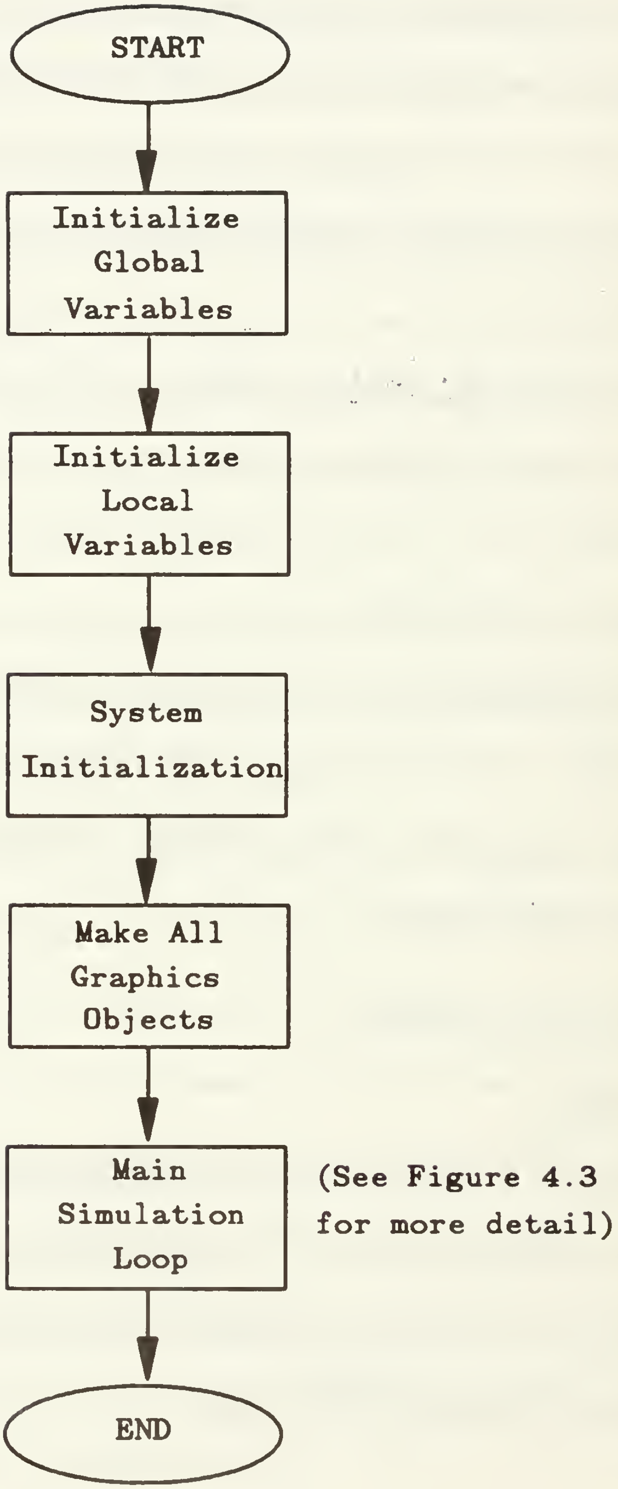

START
Initialize Global Variables
Initialize Local Variables
System Initialization
Make All Graphics Objects
Main Simulation Loop
(See Figure 4.3 for more detail)
END

Figure 4.2 CARSIMU.C Flowchart

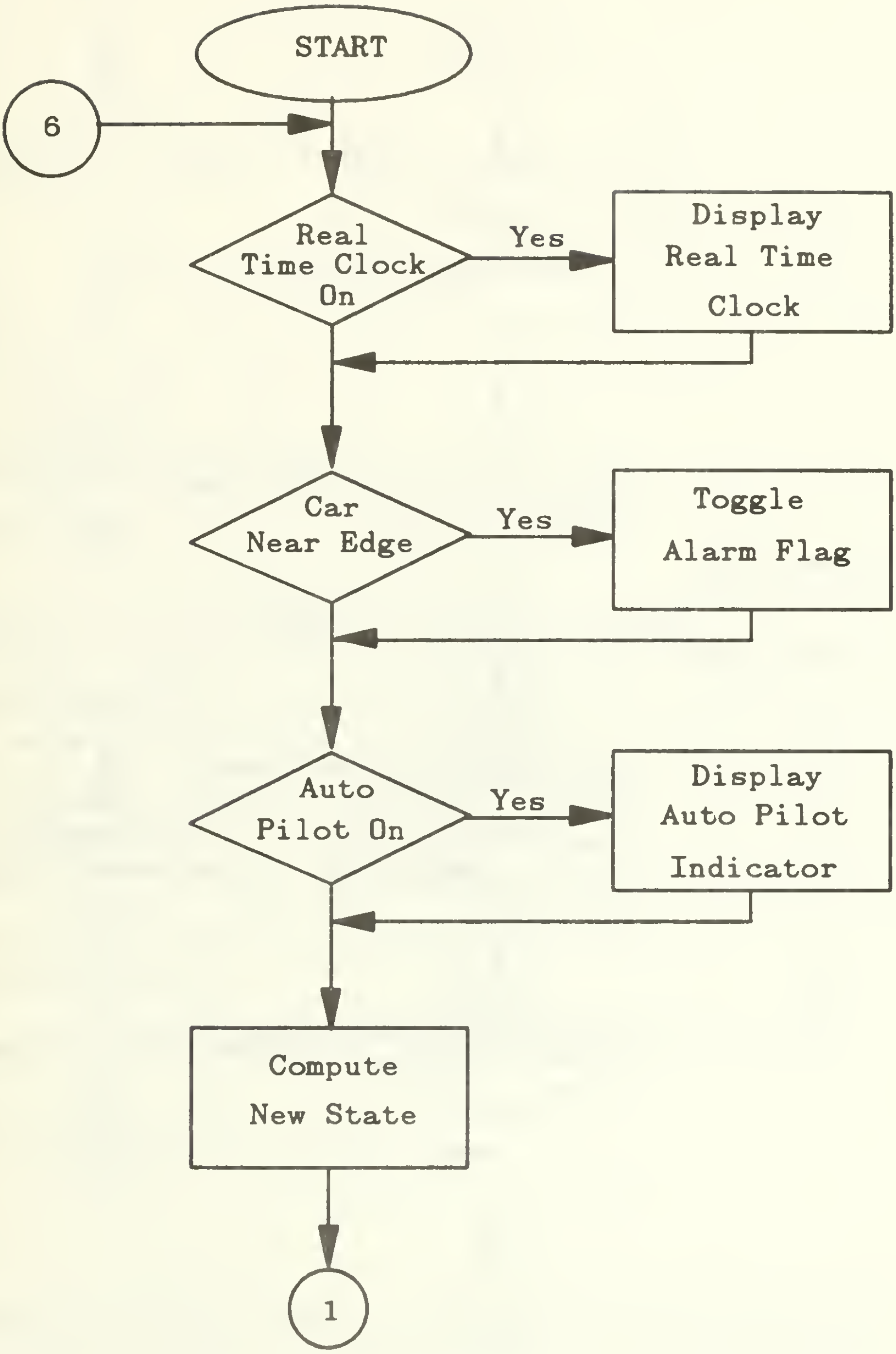


Figure 4.3 Main Simulation Loop Flowchart (Part 1)

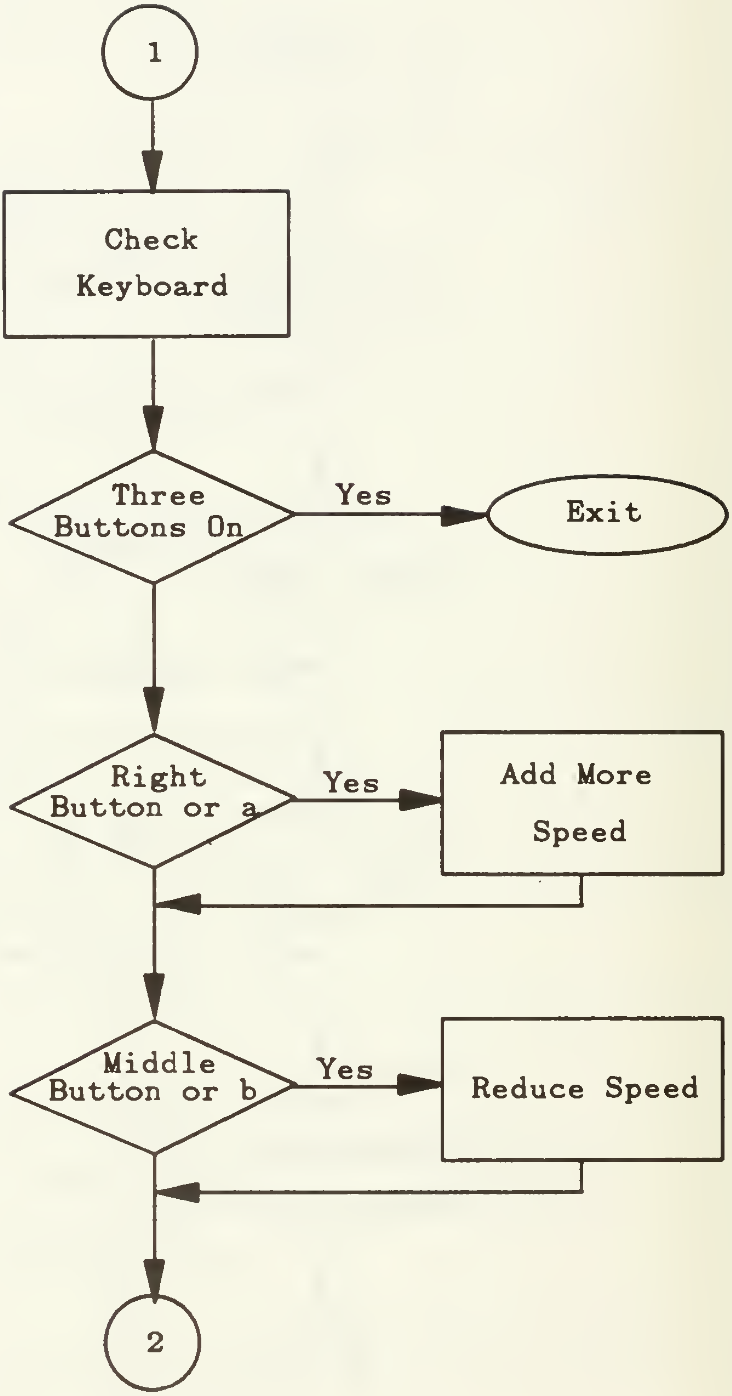


Figure 4.4 Main Simulation Loop Flowchart (Part 2)

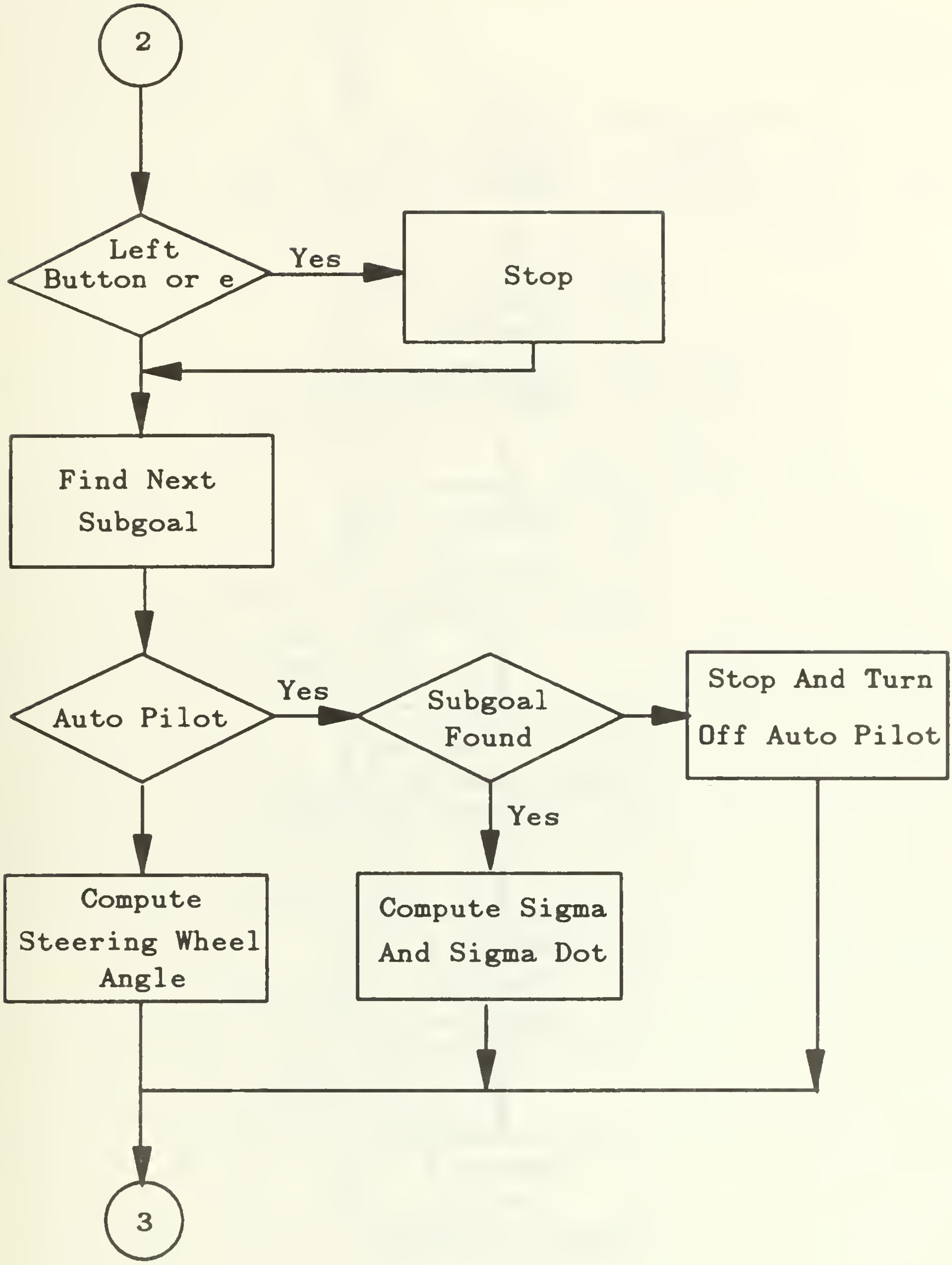

2
Left Button or e
Yes
Stop
Find Next Subgoal
Auto Pilot
Yes
Subgoal Found
Stop And Turn Off Auto Pilot
Yes
Compute Steering Wheel Angle
Compute Sigma And Sigma Dot
3

Figure 4.5 Main Simulation Loop Flowchart (Part 3)

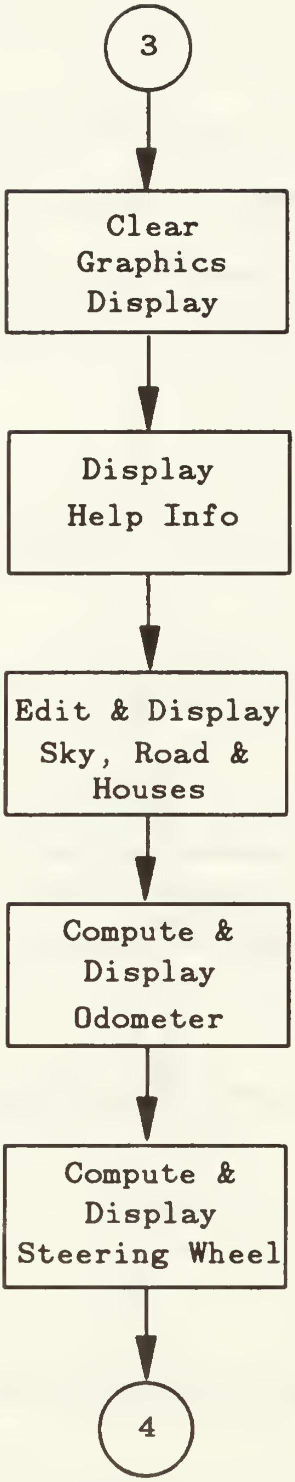

3
Clear Graphics Display
Display Help Info
Edit & Display Sky, Road & Houses
Compute & Display Odometer
Compute & Display Steering Wheel
4

Figure 4.6 Main Simulation Loop Flowchart (Part 4)

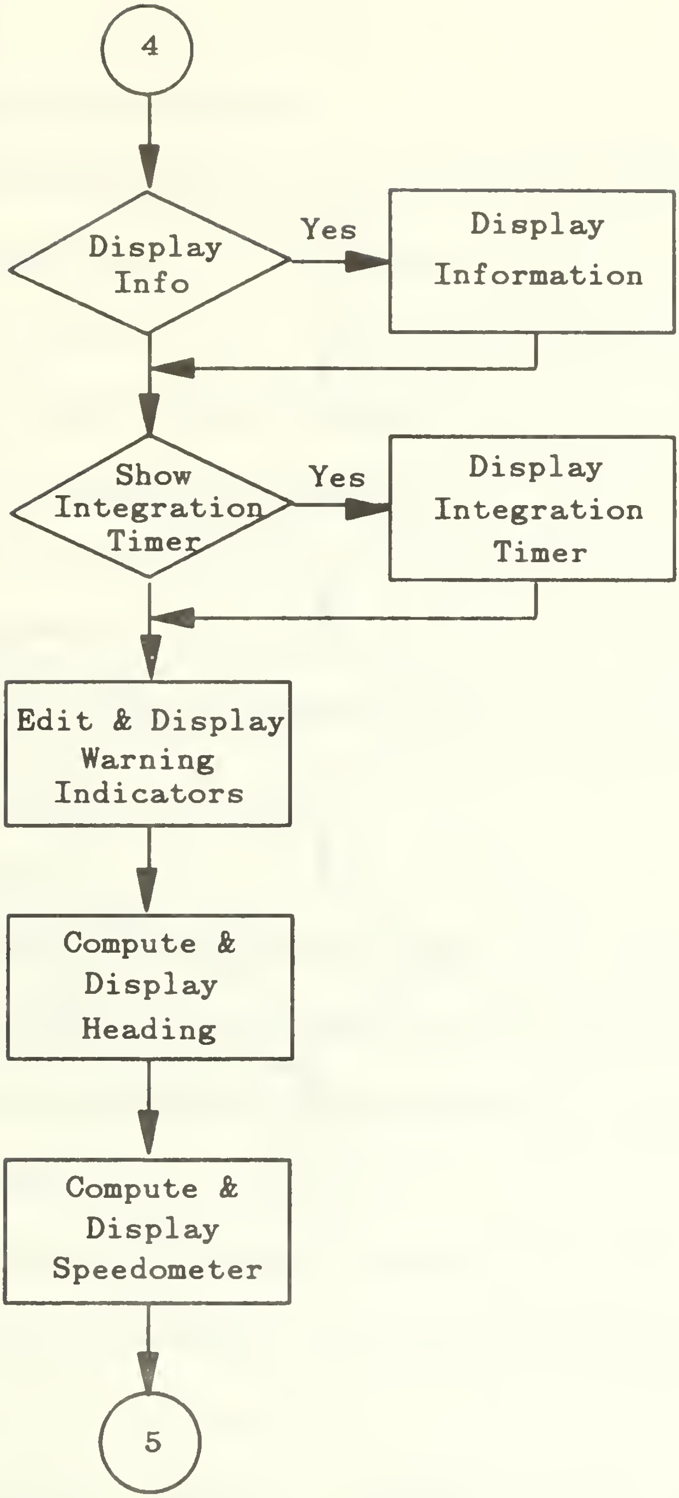

4
Display Info
Yes
Display Information
Show Integration Timer
Yes
Display Integration Timer
Edit & Display Warning Indicators
Compute & Display Heading
Compute & Display Speedometer
5

Figure 4.7 Main Simulation Loop Flowchart (Part 5)

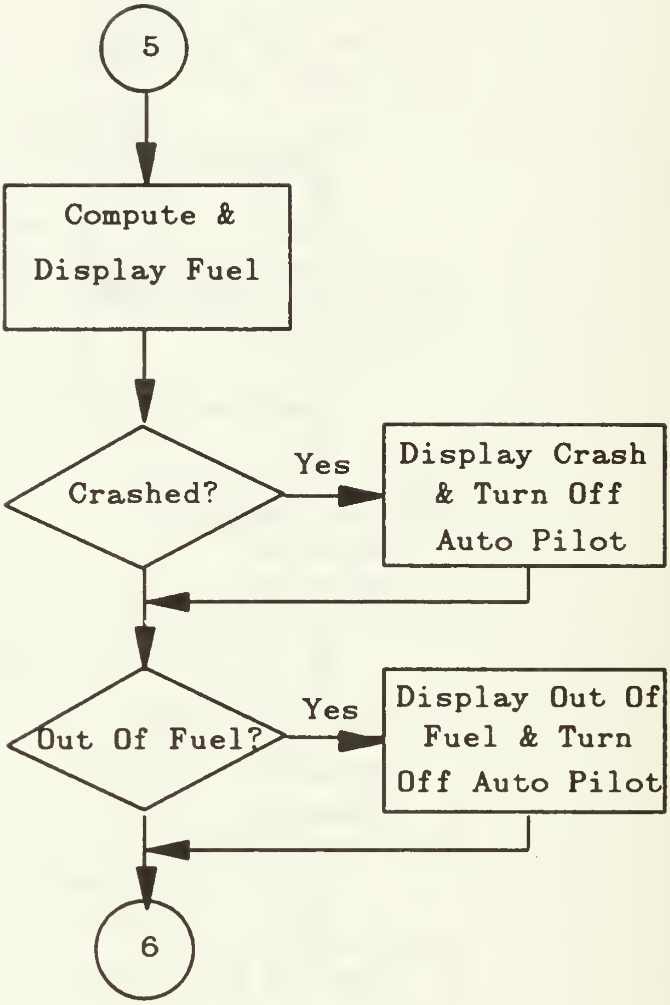


Figure 4.8 Main Simulation Loop Flowchart (Part 6)

2. Circuit.c

This module has three main functions. The first and primary function is to build the road used in the graphics simulation. The road is built with four straight segments of equal width and length. These segments are connected together by three road curves to form a open-ended rectangular circuit. The length and width of the straight segments can be varied individually. The radius of the road curves can also be individually modified.

The second function of this module is to "paint" all the road marks on the surface of the road. There are three types of road marks - white arrows, white strips and wording on the road surface. The final function is to "erect" all the signboards along the road.

3. Find-subgoal.c

The function of this module is to search for the next subgoal for the vehicle to steer towards when the vehicle is operating in the autopilot mode. This module uses the road map generated by the module map.c to compute and determine the next subgoal.

Instead of searching for the subgoal only when the vehicle is in the autopilot mode, the subgoal is constantly being computed and selected once the vehicle starts moving. There are two reasons for doing this. The first being that it provides a general solution to ensure that the subgoal is always in front of the vehicle. This is done by making sure that the very first subgoal is chosen in front of the vehicle. Subsequent subgoals are constantly recomputed and selected as the vehicle moves so that the subgoal will always remain in front of the vehicle. This simple solution works in the simulation model because the vehicle always starts from the same position and heads in the same direction.

The second reason, which is the more important one, is that of providing a subgoal quickly when the autopilot is turned on. When the subgoal is not constantly being recomputed and selected, then if the autopilot is turned on for the first time very far down the road, a significant and noticeable delay arises due to the need to search for a subgoal starting from the beginning of the road. Another situation where there is such a delay is when the autopilot is turned off and on over a long distance. In the latter situation, the subgoal has to be computed from the location where the autopilot was last turned off.

4. Map.c

This is the only module that works independently from the rest of the system. Its basic role is to generate the road map that is used for subgoal computation and selection. Though it works independently, it has to be given the same road description as that used for building the road in the main simulation module.

There are a number of road parameters that can be modified. These are the road width, road length, road curve radius and interval size. The last parameter determines how far apart road center line points are spaced in the list of points available for steering subgoals. Evidently, the smaller the interval size, the greater is the number of points generated to represent the road. For the results presented in Chapter V, the road points available for vehicle steering are stored with a spacing of 1 meter between successive points. The output of this module is a file containing the map of the entire road. This file is called *roadmap.*

5. Other.c

There are many routines in this module. Each routine builds a graphical object such as the sky, clouds and mountains. There are also a few supporting routines which are called by circuit.c to build the road used in the simulation. These supporting routines build the road surface, curves, arrows and signboards.

6. Help.c

When the key h or H is pressed, the lower half of the graphics screen that displays the vehicle dashboard is used to display help information. This module controls the content of the help information.

7. Letter.c

This routine was developed J. Artero and R. Kirsch and modified by L. Williamson. This module creates all the upper-case Roman alphabet except for G, Q, V, W, X and Z. With the graphics translate and rotate command, the appropriate letters are selected and positioned to form the wording on the surface of the road.

8. Integrate.c

The Euler-Heun numerical integration method is implemented in this module [Ref. 45]. When the vehicle is driven in manual mode, the driver controls the steering wheel with the movement of the mouse. However, when the autopilot is in control, the steering wheel angle for the dashboard display is computed.

9. Display.c

The whole dashboard display is created with this module. The vehicle dashboard is displayed on the lower half of the graphics screen. Figure 4.1 shows the various parts of the dashboard display.

10. Road.h

User defined constants in C programming are extremely important. This feature not only makes software modification easier, but also improves program readability. Besides the system defined constants that can be accessed by including the include file, gl.h, user defined constants are kept in road.h. This file also contains any additional user defined type such as *Dimension*.

11. Roadmap

This is the file generated by the module map.c. It contains the roadmap that defines the center line of the road used by the simplified vision model. Without this file, the simulation will not run.

12. Makefile

The make facility in UNIX [Ref. 42] is a very useful feature. It helps solve a lot of program administration problems associated with large software projects involving many modules. One capability that it provides is to automatically recompile only the files that have changed. The make feature is driven by an special file call Makefile. This special file defines the files that make up the entire program.

E. SPECIAL NOTES

This section highlights some of the important decisions made in the simulation program that must be understood by readers who may intend to modify and improve the simulation.

1. 45° Branch Cut

This feature is required to overcome the problem of discontinuity when the arctangent function used in the main module crosses the 180° boundary. The solution to this problem adopted in this work assumes that the maximum vehicle heading error never exceeds 45° and therefore places the arctangent branch cut at −45° rather than at the usual 180° location. This alteration permits the road center line to turn 270° without the vehicle encountering a discontinuity in σ or ψ.

2. Convention Difference

One of the issues that caused much programming difficulty initially is that the x, y, and z axis convention adopted in the mathematical model developed in Chapter III differs from that of the IRIS graphics implementation in that the graphics z axis in screen coordinates is directed inward. Anyone wishing to modify the programs of this work must be aware of this difference.

3. Roadmap Size

The size of the array used in the program for holding the roadmap, that is, the road center line, is initialized to 3000 points. This may not be sufficient for a roadmap with a smaller interval or when the present road circuit is extended. Another point to note is that the z coordinate of the road point is included so that an uneven road can be set up without much software modification. Currently, the z coordinate is set to zero.

F. SUMMARY AND CONCLUSIONS

The entire graphics simulation model is written in a manner that allows easy modification and expansion. One of the major problems with software development is to control the *rippling effect* of future code updates. Much attention was devoted to this aspect when the software system of this study was designed.

With the 3D color graphics simulation model, many experiments can be carried out involving both human and autopilot driving. The results of a number of such experiments are documented in the next chapter. The source code of the entire simulation model is included in the appendix for those who need to access to it for more detailed understanding or modification.

V. EXPERIMENTAL RESULTS

A. INTRODUCTION

Many simulation runs were carried out with different model characteristics to test the validity of the hypothesis and the correctness of the mathematical model used. The results of the 3-D simulation runs were captured, scaled, and formated into 2-D plots for documentation and discussion.

To obtain the 2-D plots, a special modification was made to the main simulation module. The purpose of the modification was to capture the vehicle position as it moves and to record its deviation from the road center line. The information is stored in two files that are also scaled. The scaling is based on the background, figure upon which this information is overlaid to obtain the desired plot.

The figures in this chapter are produced with a very flexible and easy-to-use graphics package called OZDRAW [Ref. 46]. Basically, the desired background such as the outline of the road, is generated with OZDRAW according to some scale. The information given by the modified module discussed above is then used by OZDRAW to overlay the vehicle positions onto the road outline. The combined image is then labeled and printed for documentation.

B. DESCRIPTION OF EXPERIMENTS

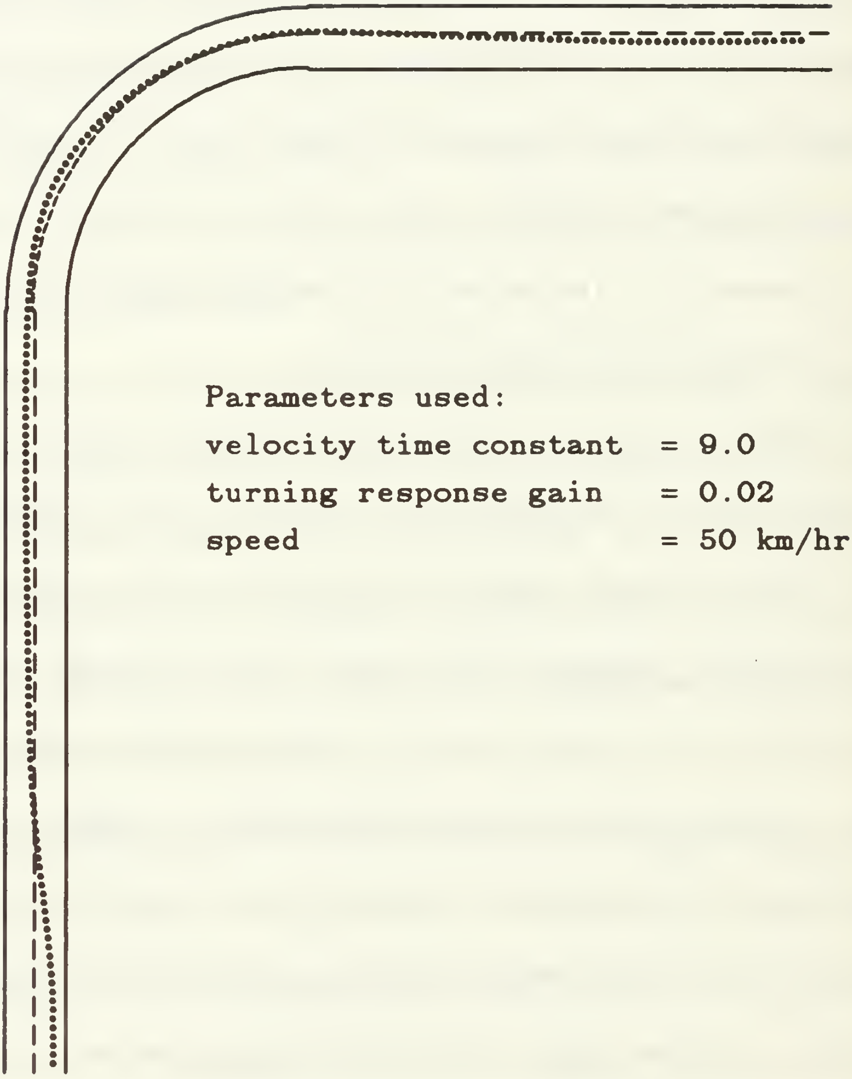
The entire simulation test track consists of four 400m straight road segments connected by three road curves with 80m radius to form an open-ended rectangular circuit. However, in most cases, not all of the circuit is used for capturing the experimental results. Rather, all but a few of the experiments were carried out for a short segment of the entire circuit which included a 200m segment of straight road followed by a road bend and then another stretch of about 150m of straight road.

In all the experiments, the simulation begins by setting the velocity to the desired value and putting the vehicle 5m off the road center line. For autopilot driving experiments, the autopilot is also activated. Except for Figure 5.3, Figure 5.4, and Figure 5.13, the simulation stops when the vehicle crashes or when it successfully overcomes the bend and it is about 150m down the second segment of the straight road. The results of Figure 5.3 and Figure 5.4 are obtained by driving the vehicle in the autopilot mode for 200m on the straight road. Figure 5.13 is generated by autopilot driving around a 270^o loop with a radius of 80m. All of the experiments use the nonlinear mathematical model for the vehicle developed in Chapter III. A comparison between the linearized model and the nonlinear model is carried out with two experiments to show their relationship and the accuracy and usefulness of the linearized model .

C. MANUAL DRIVING

When the simulation was first developed, the mouse buttons were used to manually drive the vehicle; i.e., pressing the left mouse button moved the vehicle to the left and pressing the right mouse button moved the vehicle to the right. This was found to be an uncomfortable way to drive the vehicle. The method that the simulation now uses is found to be much better. It simply involves moving the entire mouse to the left or to the right to steer the vehicle to the left or to the right respectively. This method of driving was found to be more natural and more closely resemble actual driving conditions.

Typical results for human driving are shown in Figure 5.1 and 5.2. In Figure 5.1, the vehicle velocity is 50 km/hr and at this speed human performance is clearly good. When the vehicle velocity is increased to 75 km/hr, shown in Figure 5.2, it was noted that performance deteriorated rapidly, especially when going around the road curve. In fact, control is only marginally possible. When attempting a cornering at this speed, several trials had to be carried out before a plot was obtained. Several attempts to manually drive the vehicle at 100 km/hr were made, but all were unsuccessful. It should be noted that the dots on these plots as well as all other figures of this chapter are generated at a rate of approximately 6 Hz and therefore they are more spread out at higher vehicle speeds.



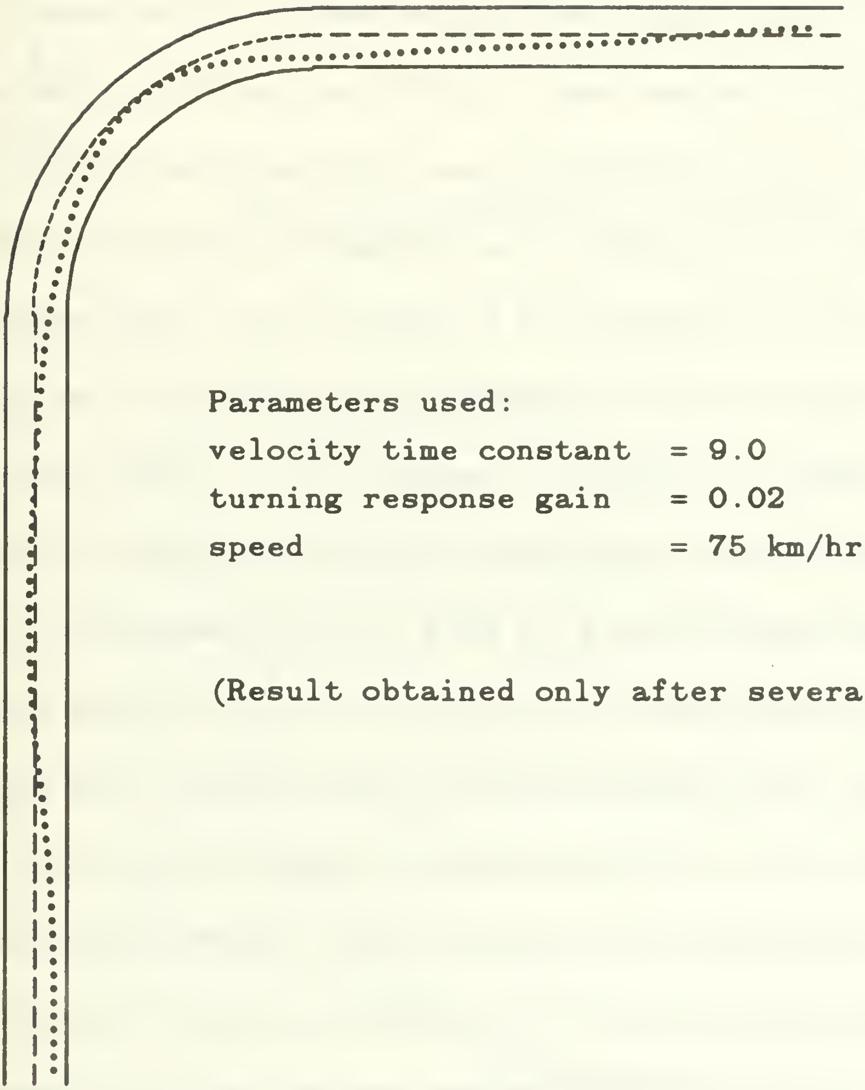
Parameters used:

velocity time constant = 9.0

turning response gain = 0.02

speed = 50 km/hr

Figure 5.1 Manual Driving



Parameters used:
velocity time constant = 9.0
turning response gain = 0.02
speed = 75 km/hr

(Result obtained only after several tries)

Figure 5.2 Manual Driving

D. COMPARISON OF LINEARIZED AND ACTUAL MODEL

The next two experiments were conducted to demonstrate the effectiveness of using linearization theory. In these experiments, the vehicle was driven by the autopilot on a 200m straight road using the pursuit navigation model. For each experiment, two runs were made. The first run used the 4th order nonlinear model of Eq. (3.10) to (3.13) with σ and $\dot{\sigma}$ calculated from Eq. (3.21) and Eq. (3.22) respectively. The second run used the linearized model where σ and $\dot{\sigma}$ were computed with the linearization of Eq. (3.24) and Eq. (3.25) respectively. The results of these two runs were overlaid to obtain Figure 5.3 and Figure 5.4. Figure 5.3 used $T = 1$ and $k_\sigma = 1$ resulting in $\lambda = -1$. This means that the system should be able to correct about 60% of its deviation from the center line in two seconds. Figure 5.4 used $T = 2$ and $k_\sigma = 0.5$ which means that $\lambda = -0.5$. In this case, the time required to correct the same amount of deviation is doubled to four seconds. These predictions match the results obtained in both figures. It is also observed that in both experiments, the linearized model used is a very accurate approximation of the nonlinear model. Moreover, the good agreement between the analytical solution and the numerical solution in both cases provides a measure of confidence that the simulation program is correct. It also shows that the sampling rate of 6 Hz adopted in this work is adequate for accurate numerical integration of the simulation model differential equations.

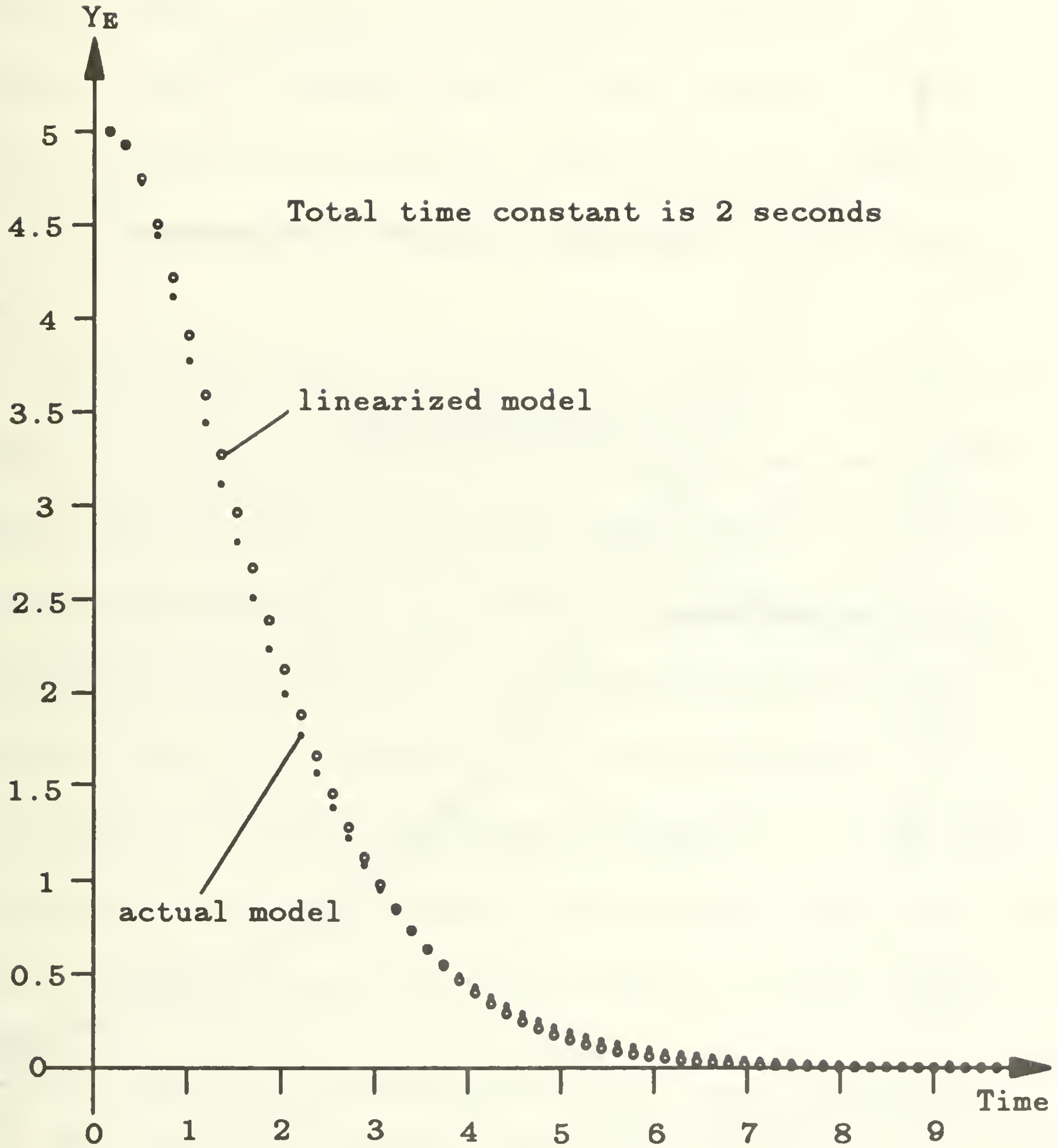


Figure 5.3 Vehicle displacement from road center line

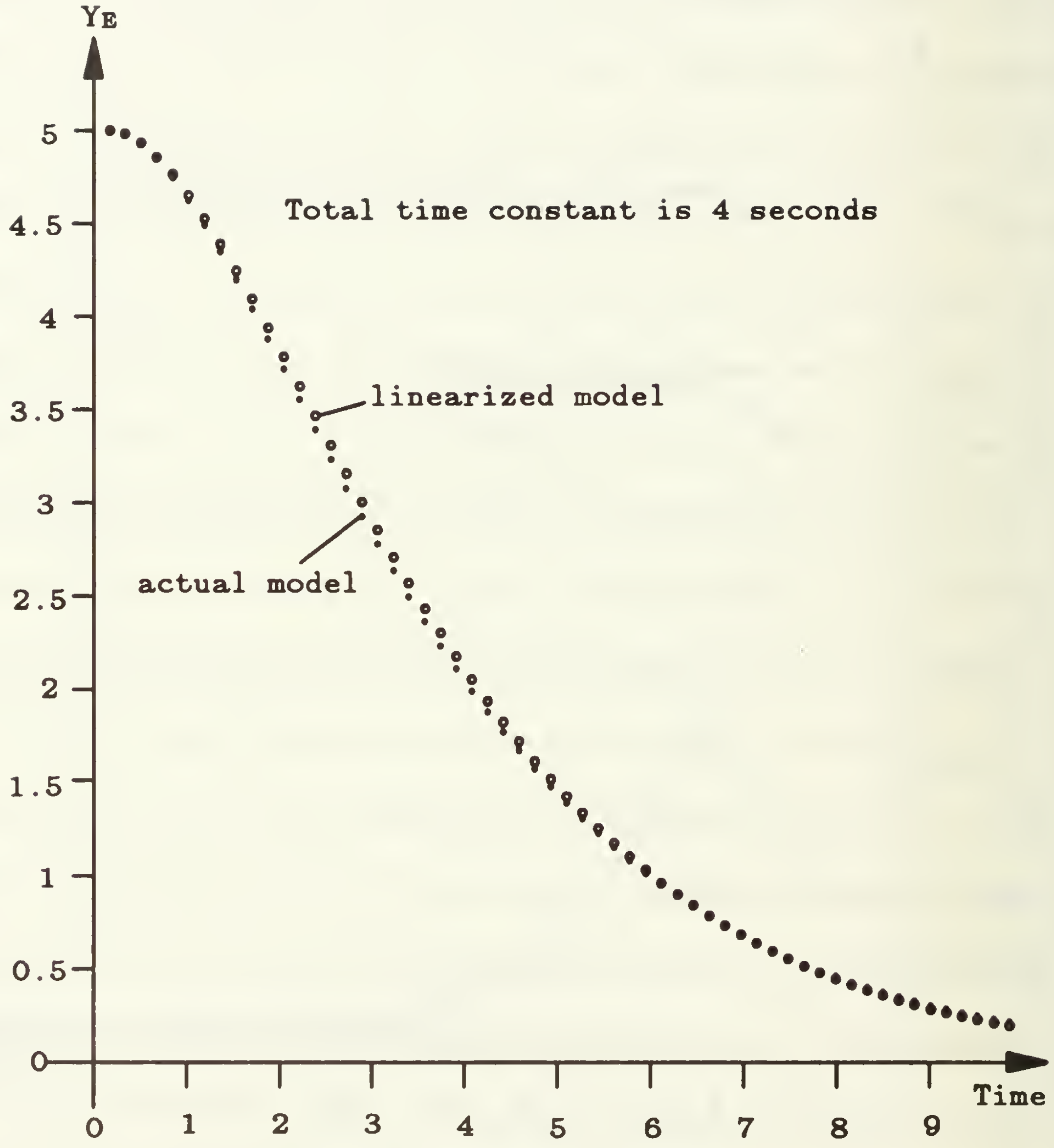
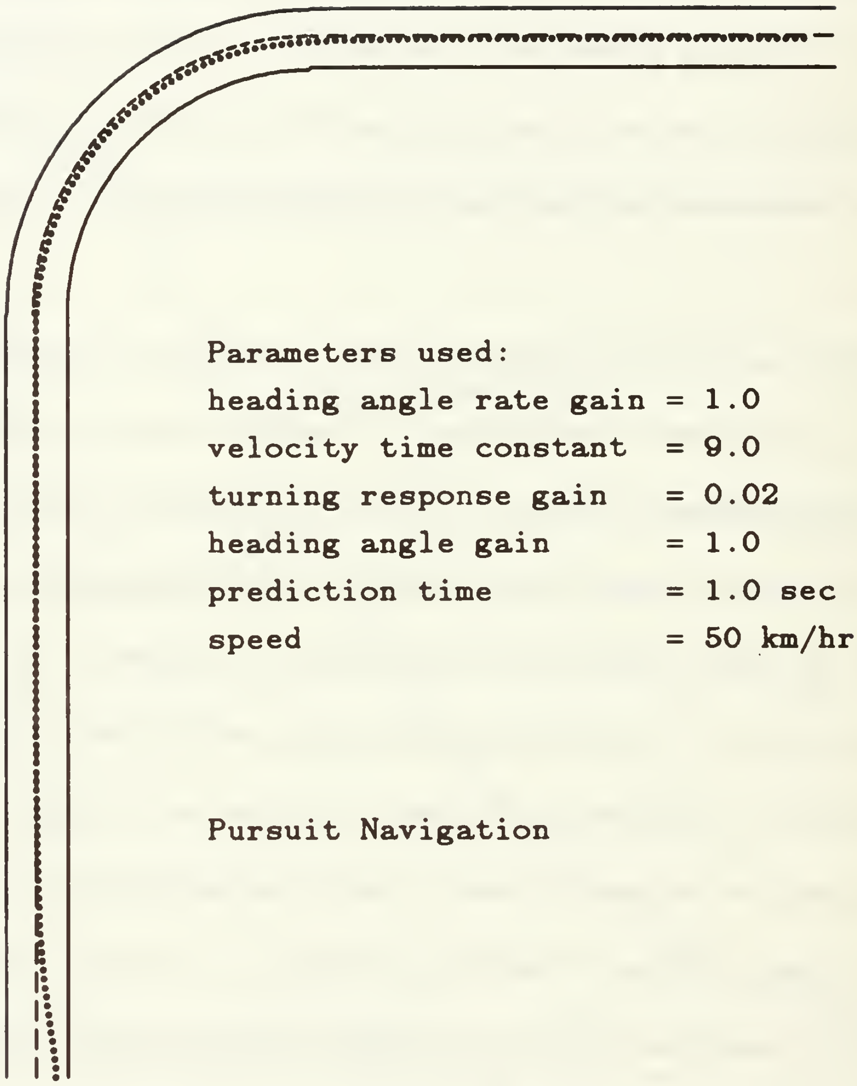


Figure 5.4 Vehicle displacement from road center line

E. PURSUIT NAVIGATION

The next three figures show the performance of the pursuit navigation model using various vehicle velocities. Figure 5.5 shows that at 50 km/hr, the vehicle was able to keep to the center line of the road quite well while turning a corner. Cornering is still not a problem when the vehicle is traveling at 100 km/hr as shown in Figure 5.6. But at 150 km/hr, Figure 5.7, the vehicle cannot keep itself on the road when turning the bend due to its inability to keep to the center of the road.

Using the same pursuit navigation model, another set of experiments was carried out. In this set of experiments, the prediction time was doubled to 2 seconds. This experiment, as shown in Figure 5.8 and Figure 5.9, shows that the autopilot performance dropped dramatically. Due to the longer prediction time, at 50 km/hr, the vehicle tends very much towards the inside of the road as compared to Figure 5.5. At 100 km/hr, the vehicle could not make it around the bend because it is predicting too far ahead. However on a straight road, the prediction time does not seem to matter. This corresponds closely to the author's concept of human driving. When we are driving on a straight road, we are normally more casual than when we are trying to bring the vehicle around a road bend; that is, we appear to shorten or lengthen our prediction time according to the road conditions and the driving environment instead of utilizing a constant prediction time as in this simulation.

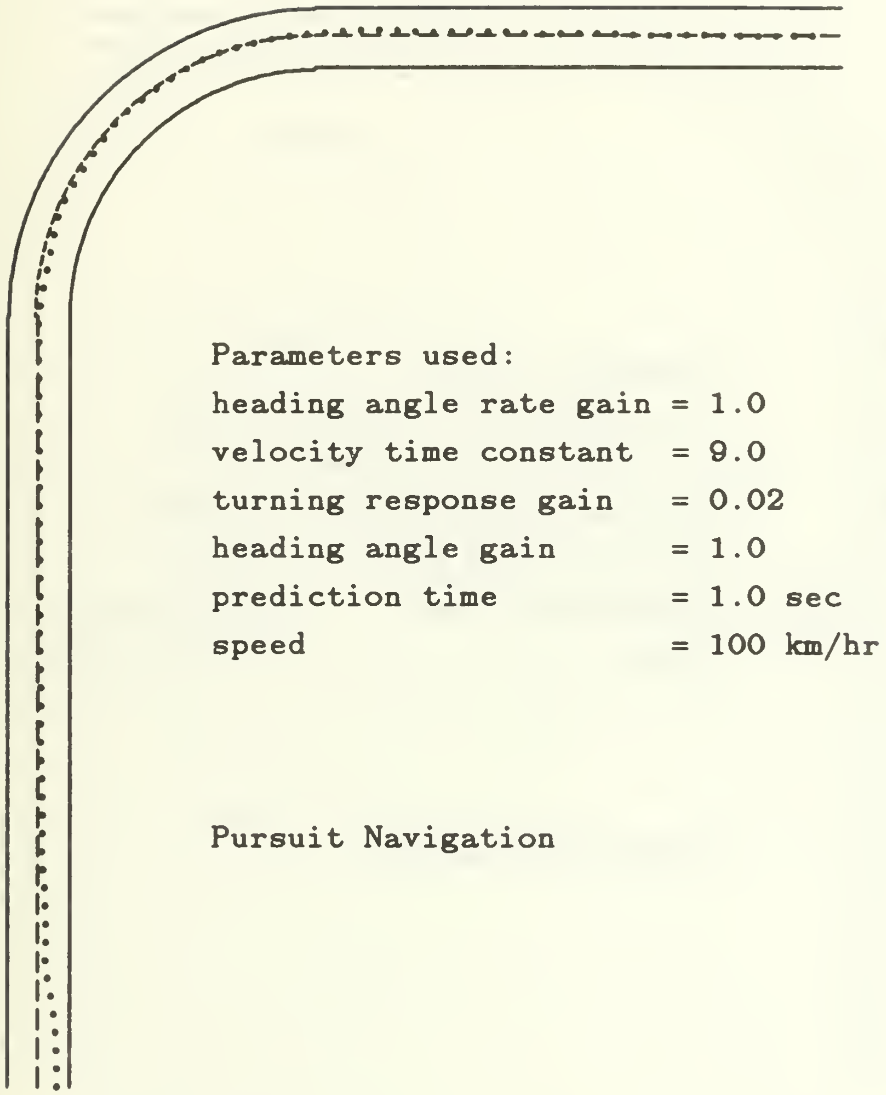


Parameters used:

heading angle rate gain	=	1.0
velocity time constant	=	9.0
turning response gain	=	0.02
heading angle gain	=	1.0
prediction time	=	1.0 sec
speed	=	50 km/hr

Pursuit Navigation

Figure 5.5 Driving with autopilot

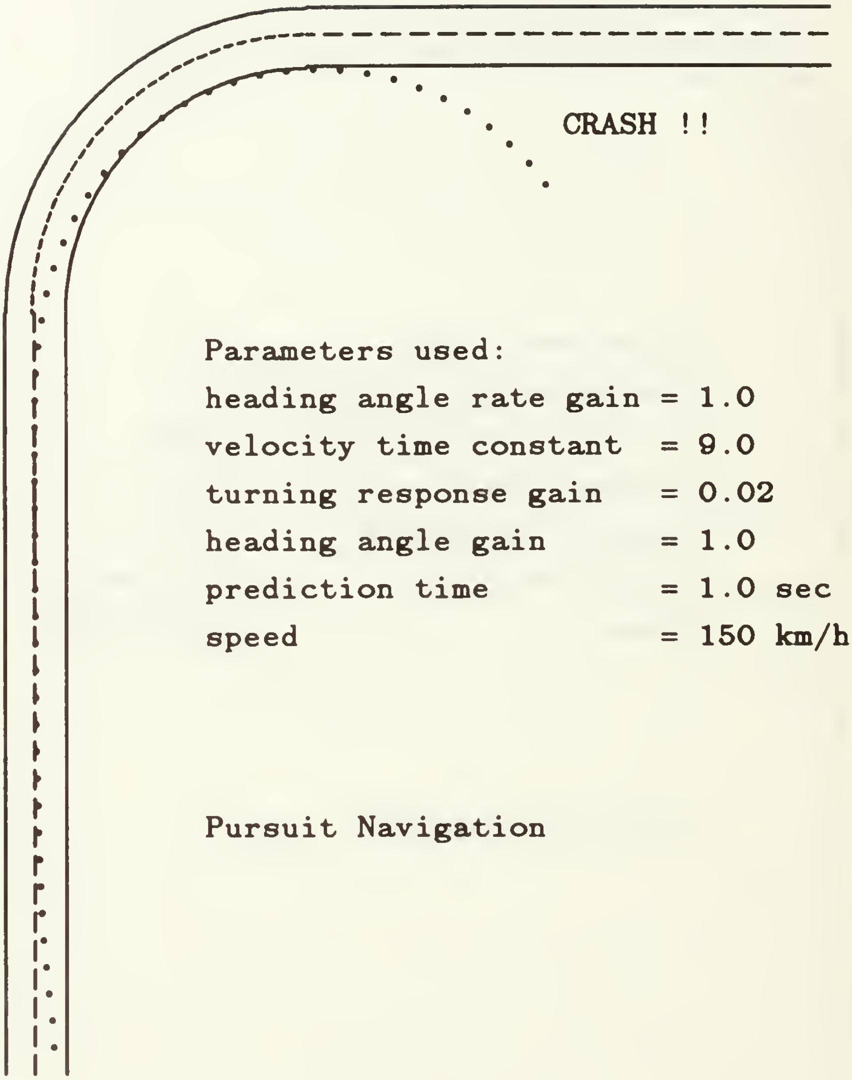


Parameters used:

heading angle rate gain	= 1.0
velocity time constant	= 9.0
turning response gain	= 0.02
heading angle gain	= 1.0
prediction time	= 1.0 sec
speed	= 100 km/hr

Pursuit Navigation

Figure 5.6 Driving with autopilot



Parameters used:

heading angle rate gain = 1.0

velocity time constant = 9.0

turning response gain = 0.02

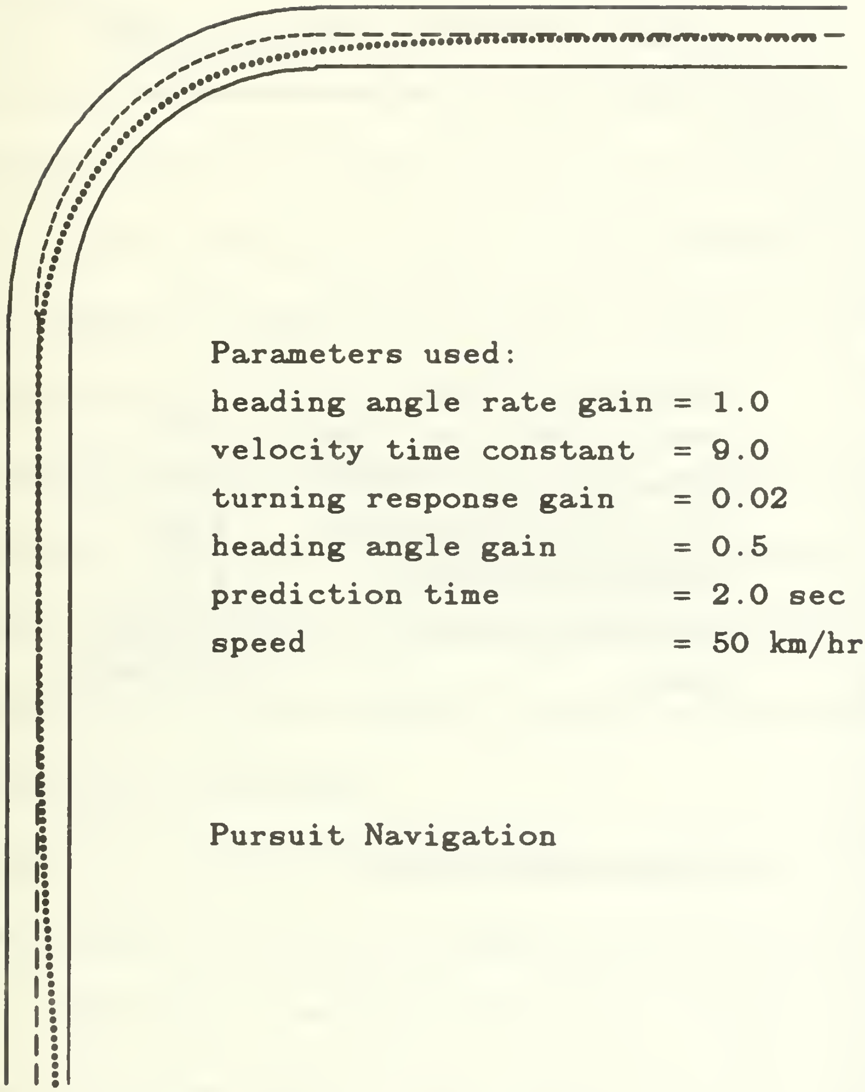
heading angle gain = 1.0

prediction time = 1.0 sec

speed = 150 km/hr

Pursuit Navigation

Figure 5.7 Driving with autopilot



Parameters used:

heading angle rate gain = 1.0

velocity time constant = 9.0

turning response gain = 0.02

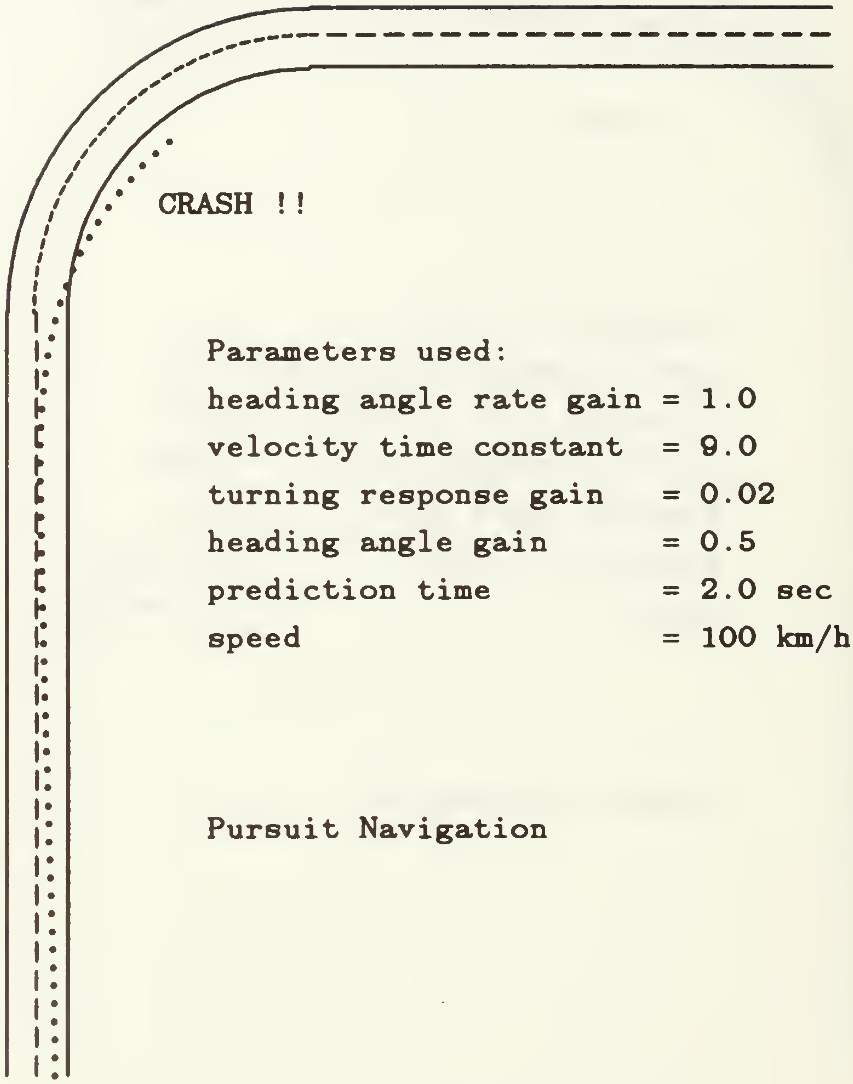
heading angle gain = 0.5

prediction time = 2.0 sec

speed = 50 km/hr

Pursuit Navigation

Figure 5.8 Driving with autopilot



Parameters used:

heading angle rate gain = 1.0
velocity time constant = 9.0
turning response gain = 0.02
heading angle gain = 0.5
prediction time = 2.0 sec
speed = 100 km/hr

Pursuit Navigation

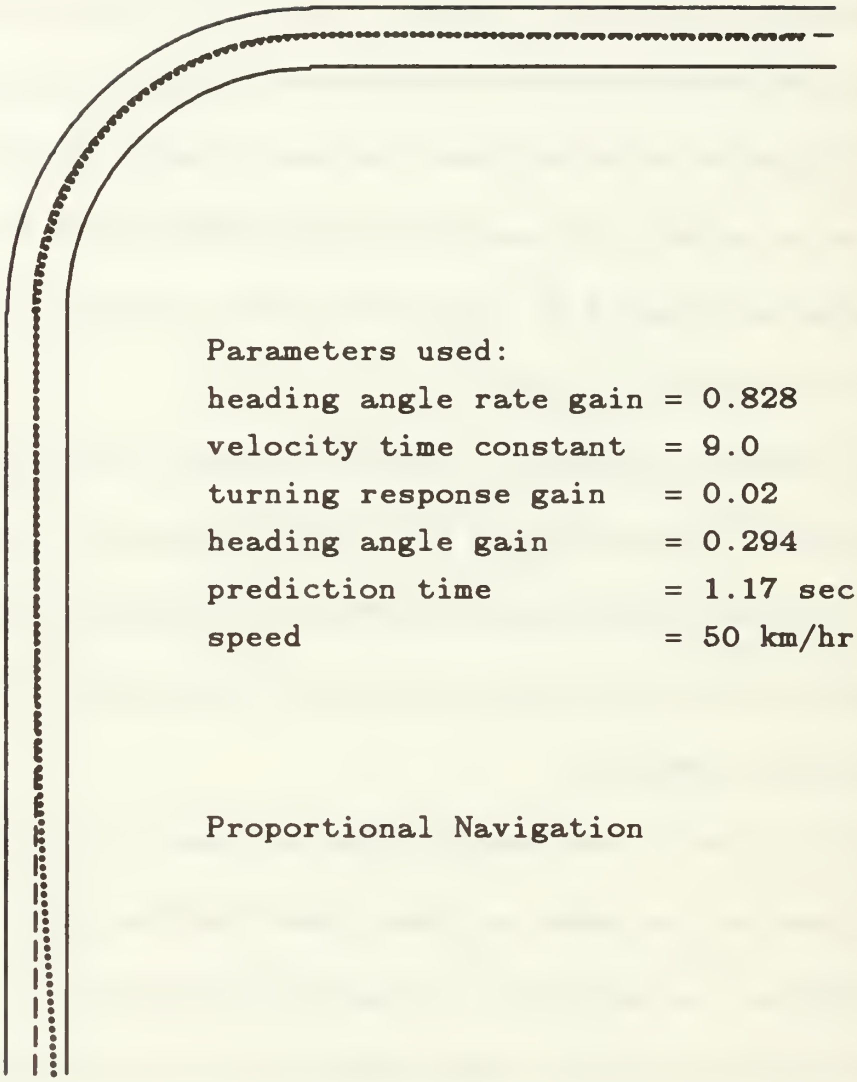
Figure 5.9 Driving with autopilot

F. PROPORTIONAL NAVIGATION

The pursuit navigation model was improved with another gain term added to the vehicle guidance law. This made it into a proportional navigation model whose performance is illustrated by Figures 5.10 through 5.12. Comparing Figure 5.10 to Figure 5.5, it can be seen that the proportional navigation model was able to maintain its position on the road center line more diligently. The importance of this is that the vehicle is now capable of turning the road bend at 150 km/hr without crashing as in Figure 5.12. This was not possible at the same speed with the pursuit navigation model, Figure 5.7.

Another important observation is the system's response to deviation. Based on results derived in Chapter III, Figure 5.5 to Figure 5.7 should have a total time constant of 2 seconds whereas Figure 5.8 to Figure 5.12, Figure 5.15 and Figure 5.16 should have a total time constant of 4 seconds. As seen in these figures, they match the predicted preformance.

Figure 5.13 shows a performance comparison between the proportional navigation model and the pursuit navigation model when the vehicle is going around a 270^o loop. As predicted by the analysis of Chapter III, the pursuit navigation steering law produces a steady displacement of the vehicle toward the inside of the turn. This effect is eliminated when proportional navigation is used with $k_{\dot{\sigma}} = 0.828$, again as predicted by linearized system analysis.

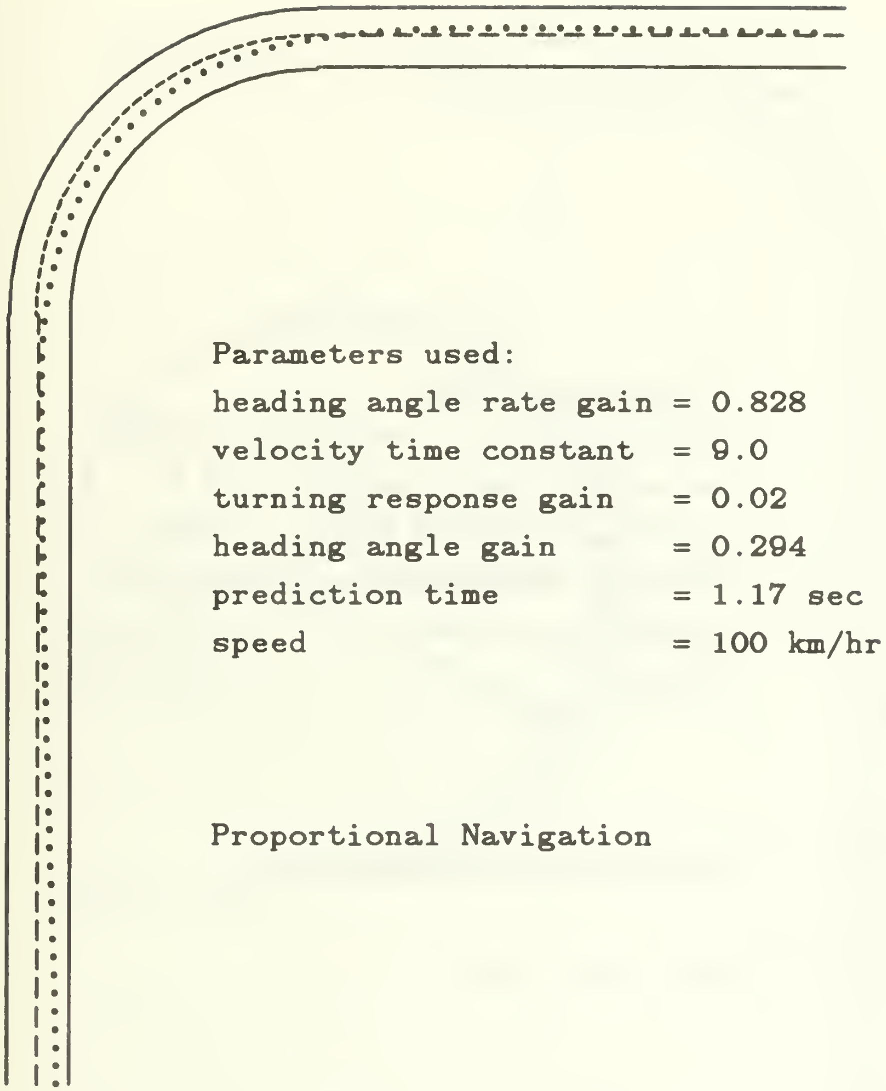


Parameters used:

heading angle rate gain	= 0.828
velocity time constant	= 9.0
turning response gain	= 0.02
heading angle gain	= 0.294
prediction time	= 1.17 sec
speed	= 50 km/hr

Proportional Navigation

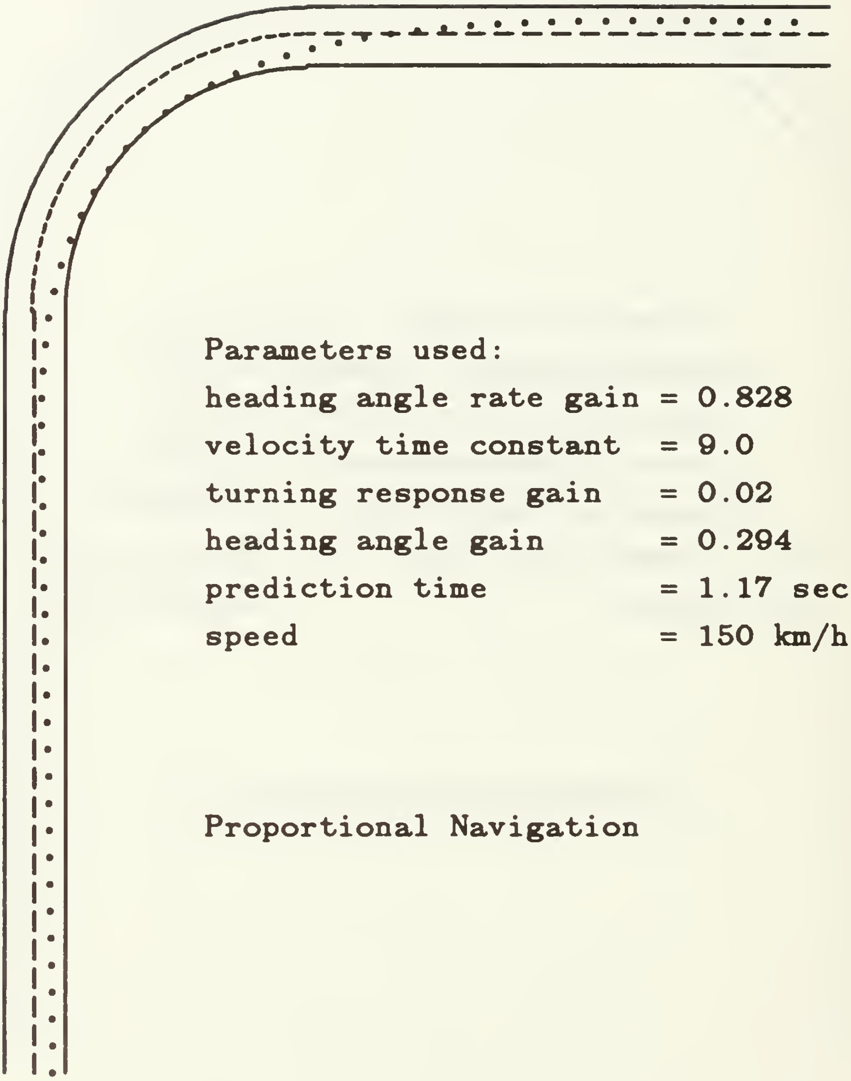
Figure 5.10 Driving with autopilot



Parameters used:
heading angle rate gain = 0.828
velocity time constant = 9.0
turning response gain = 0.02
heading angle gain = 0.294
prediction time = 1.17 sec
speed = 100 km/hr

Proportional Navigation

Figure 5.11 Driving with autopilot



Parameters used:
heading angle rate gain = 0.828
velocity time constant = 9.0
turning response gain = 0.02
heading angle gain = 0.294
prediction time = 1.17 sec
speed = 150 km/hr

Proportional Navigation

Figure 5.12 Driving with autopilot

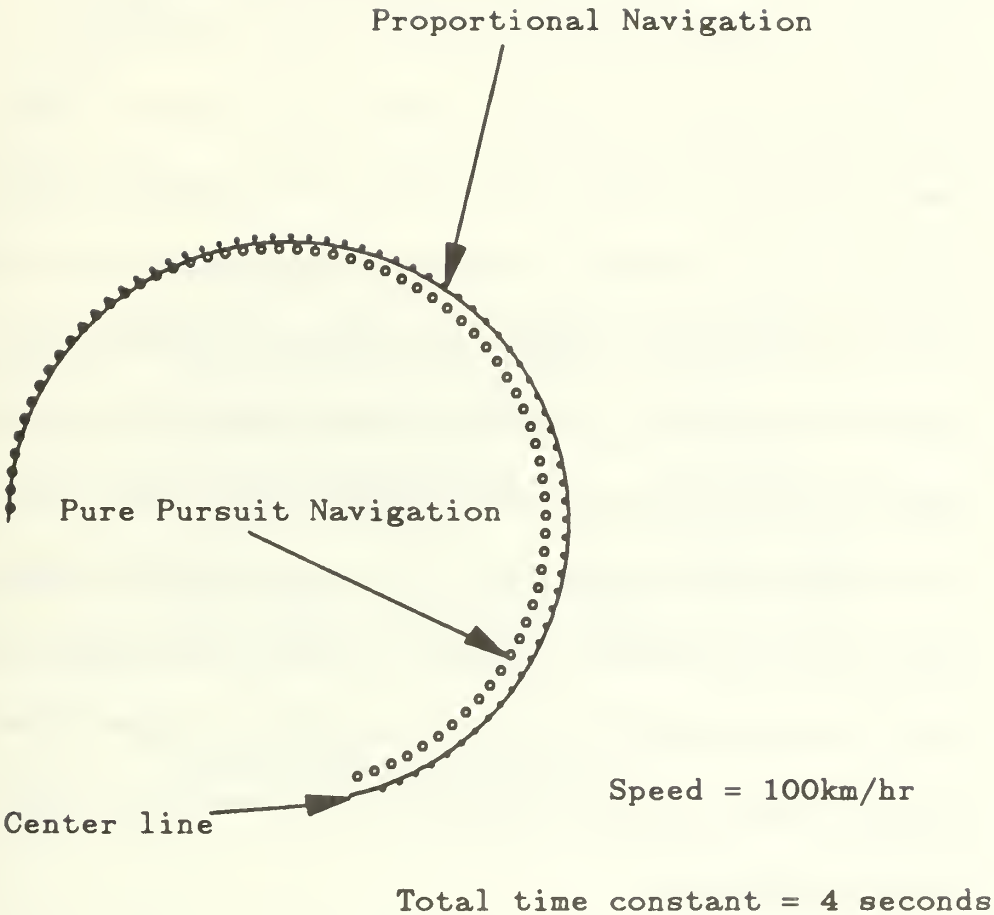
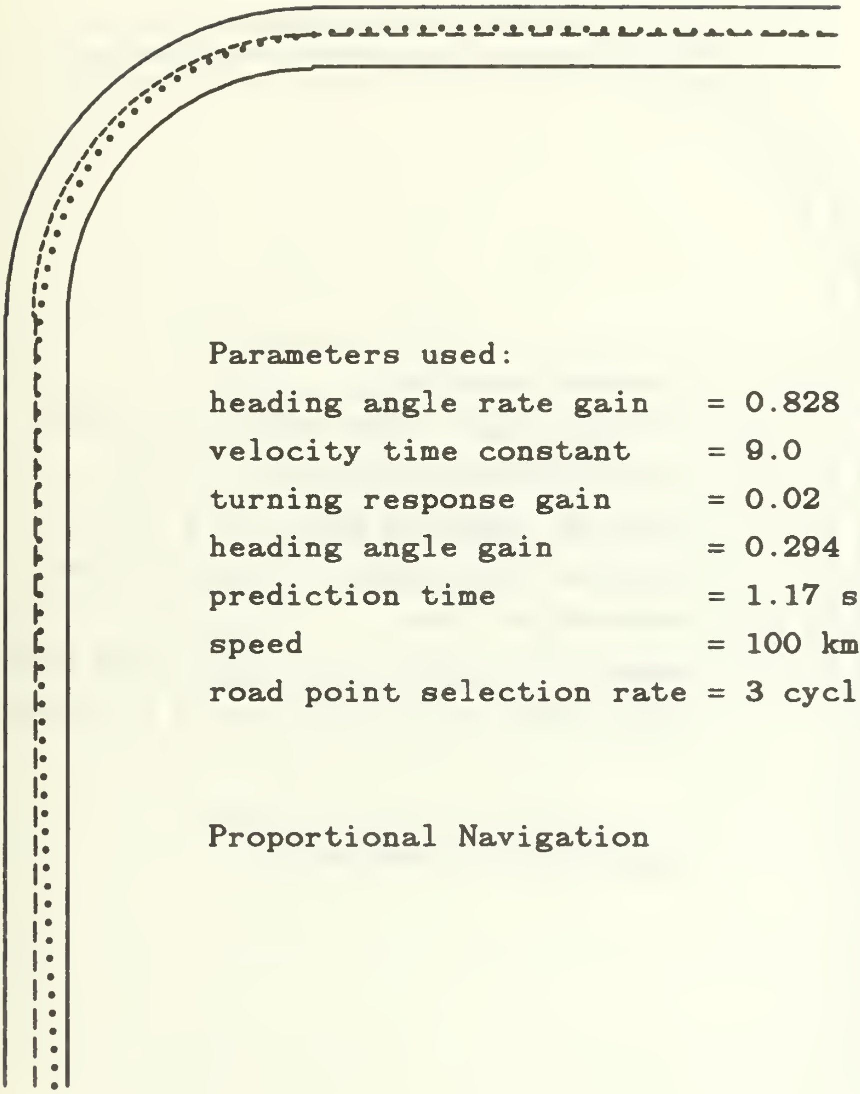


Figure 5.13 Loop Performance

G. EFFECT OF VARIOUS ROAD POINT SAMPLING RATES

The last set of experiments carried out was to examine the effect of different sampling frequencies at which road steering points are selected when the vehicle is operating in autopilot mode. All the autopilot driving experiments conducted previously were done with the steering point being selected every cycle. Figures 5.14 through Figure 5.16 show the results obtains when new steering points are chosen after every 3 cycles, 5 cycles and 7 cycles respectively with a 1.17 second prediction time. Comparison of these figures with Figure 5.14 shows that choosing steering points less often actually helps the vehicle to negotiate curves. This may seem surprising initially, but in fact should be expected since reducing the road sampling rate also reduces the *average* prediction time. There is of course a limit to the extent that the sampling rate can be lowered since the steering point must not be allowed to pass under the vehicle. For the present example, since $T = 1.17$ seconds, the maximum interval for road sampling is 7 control cycles (because, as previously stated, the vehicle steering loop was operated by a 6 MHz rate for all simulation experiments).

While the above analysis seems to say that the apparent human strategy of driving toward one point for several steering cycles is effective, one negative result of this approach was noted. This was that the steering wheel motion was more jerky than in earlier simulations in which a new steering point was selected on every control cycle. Again, this is not surprising since reducing the average

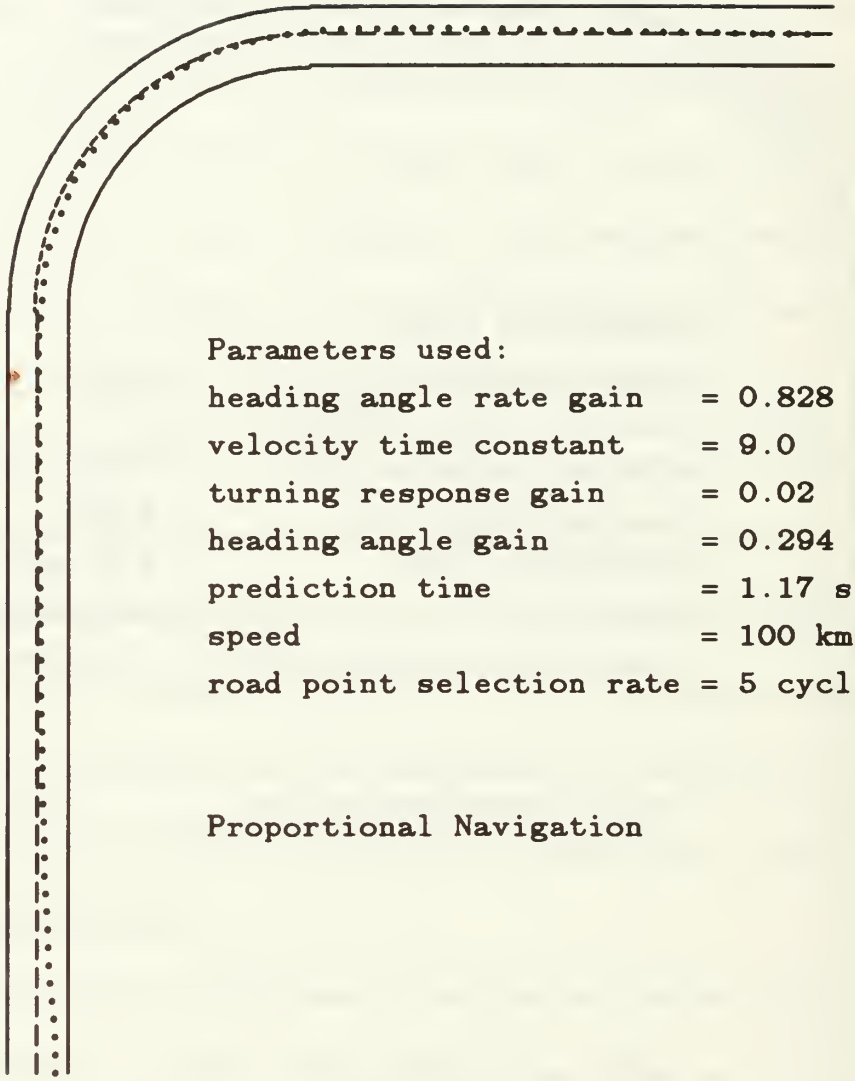


Parameters used:

heading angle rate gain	=	0.828
velocity time constant	=	9.0
turning response gain	=	0.02
heading angle gain	=	0.294
prediction time	=	1.17 sec
speed	=	100 km/hr
road point selection rate	=	3 cycles

Proportional Navigation

Figure 5.14 Driving with autopilot

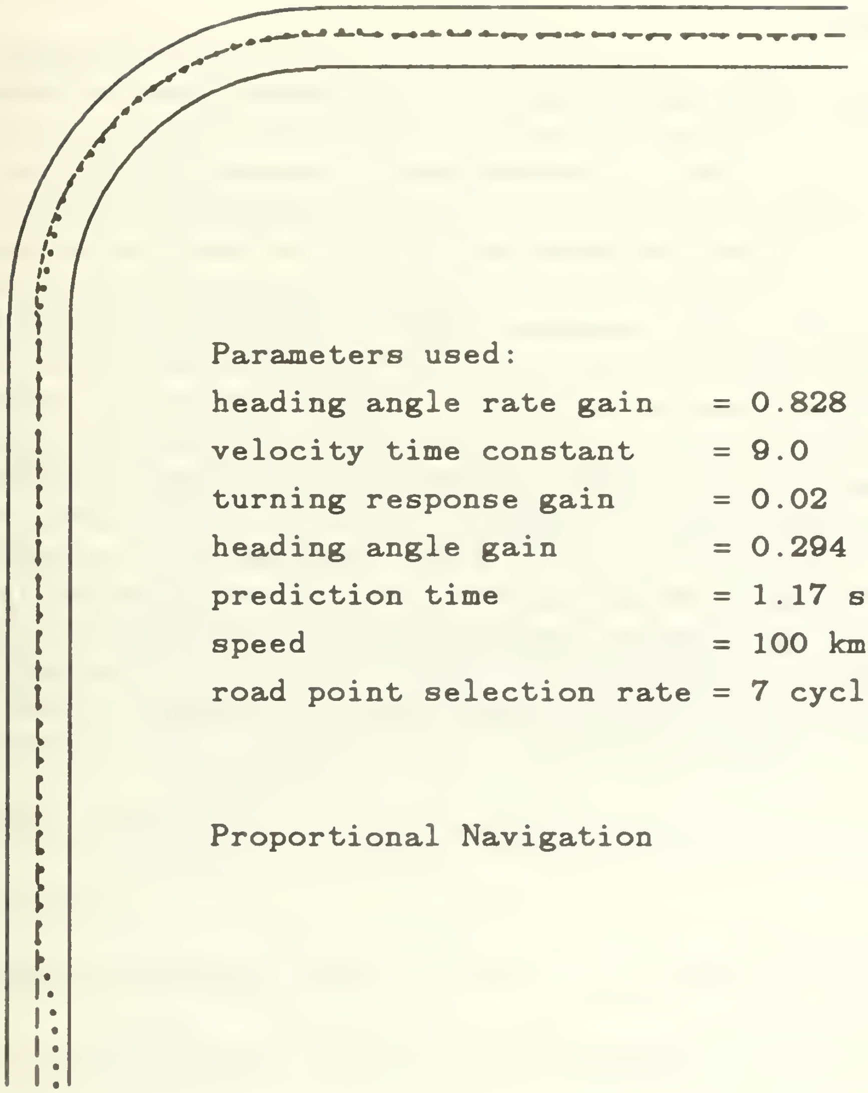


Parameters used:

heading angle rate gain	=	0.828
velocity time constant	=	9.0
turning response gain	=	0.02
heading angle gain	=	0.294
prediction time	=	1.17 sec
speed	=	100 km/hr
road point selection rate	=	5 cycles

Proportional Navigation

Figure 5.15 Driving with autopilot



prediction time should tend to cause the system to become somewhat underdamped according to the theory of Chapter III.

H. SUMMARY

The results in this chapter show that the hypothesis about the unconscious behavior of human driving is reasonably well in agreement with reality. It also shows that the mathematical model developed in the earlier chapter is a useful model for mimicking this unconscious behavior. However, much more work is needed to obtain statistical information about how human gain values and prediction times vary with driving conditions before any degree of confidence can be attached to the hypothesis of this work. Regardless of the results of such a study, however, the hypothesis used for unconscious human behavior in steering clearly provides a viable basis for autopilot design for autonomous vehicles.

VI. SUMMARY AND CONCLUSIONS

A. SUMMARY

This research work differs from most previous research work in the area of lateral control of autonomous vehicles in the sense that steering laws have not generally been derived explicitly from a model of human task performance. Rather, much of the current research focuses mainly on vision, sensors, planning, navigation, and obstacle avoidance. So far as the author knows, none has explored the behavioral aspects of human driving which could provide some different insights into possible approaches to autonomous navigation.

As observed at the start of this work, human driving can be divided into two distinct levels: that of conscious and unconscious behavior. This work is concerned entirely with studying and modeling of the unconscious aspect of human driving.

Another important product of this work is the development of a 3-D color graphics simulation model. This model can be modified, enlarged and enhanced to incorporate other related research work in the future. With this model, the experiments are more interesting and realistic than using simple 2-D data plots as has been done in many previous simulation studies.

B. CONCLUSIONS AND POSSIBLE EXTENSIONS

In this work, all the different subsystems such as the vision subsystem, the vehicle control subsystem, etc., are treated all together as one system. This is manageable because of the various simplifying assumptions made in the mathematical model for vehicle and driver behavior. One direction to enlarge this research work is to develop a more sophisticated vision model. The present vision mechanism is too simple and assumes perfect vision capable of "seeing" a point down the road for the vehicle to steer towards. A better model could use a more elaborate algorithm and techniques such as texture and color analysis to determine the various road features and to estimate the road edge location.

Another possible extension to this work is to study the conscious aspect of human driving. An example of this conscious behavior would be the ability to stop the vehicle appropriately when the vision subsystem "sees" a stop sign along the road or an obstacle large enough to prevent the vehicle from going ahead further.

In conclusion, the author hopes that this research work can serve as a testbed and motivation for a more elaborate and comprehensive study into the behavioral aspects of human driving. This could have a significant impact on the development of viable autonomous vehicles in the future.

LIST OF REFERENCES

1. Coiffet, Philippe, *Robots Technology Volume 1: Modelling and Control*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

2. Coiffet, Philippe, *Robots Technology Volume 2: Interaction with the Environment*, Prentice-Hall. Inc., Englewood Cliffs, New Jersey, 1983.

3. Warring, R.H., *Robots & Technology*, Tab Books Inc, Blue Ridge Summit, Pennsylvania, 1984.

4. Horgan, John, "Roboticists Aim to Ape Nature," *IEEE Spectrum*, v. 23, n. 2, pp. 66-71, February 1986.

5. Adam, John A., "Aerospace and Military," *IEEE Spectrum*, v. 23, n. 1, pp. 76-81. January 1986.

6. Albus, James S., *Brains, Behavior, & Robotics*, BYTE Publications, Inc., New York, 1981.

7. Thring, M.W., *Robots and Telechirs*, John Wiley & Sons, New York, 1983.

8. McGhee, Robert B., "Vehicular Legged Locomotion," in *Advances in Automation and Robotics*, edited by G. N. Saridis, JAI Press, Inc., Greenwich, Connecticut, 1985.

9. Raibert, Marc H., *Legged Robots that Balance*, The MIT Press, Cambridge, Massachusetts, 1986.

10. Waldron K.J. and McGhee, R.B., "The Adaptive Suspension Vehicle," *IEEE Control Systems Magazine*, v. 6, n. 6, pp. 7-12, December 1986.

11. Orin, D.E., "Supervisory Control of a Multilegged Robot," *Int. J. Robotics Res.*, v. 1, n. 1, pp. 79-91, 1982.

12. Gurfinkel, V.S. et al., "Walking Robot with Supervisory Control," *Mechanism and Machine Theory*, n. 16, pp. 31-36, 1981.

13. Nilsson, Nils J., "A Mobile Automaton: An Application of Artificial Intelligence Techniques," *Proc. of International Joint Conferenee on Artifical Intelligence*, pp. 509-519, 1969.

14. Tenenbaum, J.M., "On Locating Objects by Their Distinguishing Features," *Computer Graphics and Image Processing*, v. 2, 1973.

15. Brooks, R.A., *Symbolic Reasoning Among 3-D Models and 2-D Images*, Stanford Artificial Intelligence Laboratory Memo AIM-343, June 1981.

16. Dongarra, Jack J., "A Survey of High-Performance Computers," *IEEE COMPCON*, pp. 8-11, March 1986.

17. Hillis, W. Daniel, *The Connection Machine*, The MIT Press, Cambridge, Massachusetts, 1985.

18. Cuadrado, John L. and Cuadrado, Clara Y., "AI in Computer Vision," *BYTE*, v. 11, n. 1, pp. 237-258, January 1986.

19. Minsky, Marvin, *A Framework For Representing Knowledge*, MIT AI Memo, Cambridge, Massachusetts, 1974.

20. Coles, L.S., "An On-Line Question-Answering System with Natural Language and Pictorial Input," *ACM Conferenee Proceedings*, 1968.

21. Coles, L.S., "Talking with a Robot in English," *Proceedings of the International Joint Conference on Artifical Intelligence*, May 1969.

22. Moravec, Hans P., *Robot Rover Visual Navigation*, UMI Research Press, Ann Arbor, Michigan, 1981.

23. Giralt, G., Chatila, R. and Vaisset, M., "An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots," *Roboties Research*, The MIT Press, Cambridge, Massachusetts, pp. 191-214, 1984.

24. Everett, Hobart R., *A Microprocessor Controlled Autonomous Sentry Robot*, Master's Thesis, Naval Postgraduate School, Monterey, California, October 1982.

25. Everett, Hobart R., "A Second Generation Autonomous Sentry Robot," *Robotics Age*, April 1985.

26. Nitao, John J. and Parodi, Alexandre M., "A Real-Time Reflexive Pilot for an Autonomous Land Vehicle," *IEEE Control Systems Magazine*, pp. 14-23, February 1986.

27. Parodi, Alexandre M., "A Route Planning System for an Autonomous System," *Proc. of 1st IEEE Conference on Artificial Intelligence Applications*, pp. 51-56, January 1984.

28. Keirsey, D., Mitchell, J., Bullock, B., Nussmeier T., and Tseng, D., "Autonomous Vehicle Control Using AI Techniques," *IEEE Trans. on Software Engineering*, v. SE-11, n. 9, pp. 986-992, September 1985.

29. Lowrie, J.. *The Autonomous Land Vehicle Program*, Martin Marietta Denver Aerospace. Denver. Colorado, December 1985.

30. Lowrie, J.. *The Autonomous Land Vehicle 1st Quarterly Report*, Martin Marietta Denver Aerospace, Denver, Colorado, May 1986.

31. Schwarzenbach J. and Gill, K.F., *System Modelling and Control*, John Wiley & Sons, New York, 1978.

32. Barnacle, H.E., *Mechanics of Automobiles*, The Macmillan Company, New York, 1964.

33. Fenton, R.E., Melocik G.C., and Olson, K.W., "On the Steering of Automated Vehicles: Theory and Experiment," *IEEE Transactions on Automatic Control*, v. AC-21, n. 3, June 1976.

34. Frank A.A. and McGhee, R.B., "Some Considerations Relating to the Design of Autopilots for Legged Vehicles," *Journal of Terramechanics*, v. 6, n. 1, pp. 23-35, 1969.

35. Fenton R.E. and Chu, P.M., "On Vehicle Automatic Longitudinal Control," *Transportation Science*, v. 11, n. 1, February 1977.

36. Fenton R.E. and Hauksdottir, A.S., "On the Design of a Vehicle Longitudinal Controller," *IEEE Transactions on Vehicular Technology*, v. VT-34, n. 4, November 1985.

37. Brogan, William L., *Modern Control Theory*, Quantum Publishers, Inc., New York, 1974.

38. McGhee, Robert B., "An Approach to Computer Coordination of Motion for Energy-Efficient Walking Machines," *Bull. Mech. Engr. Lab.*, n. 43, pp. 1-21, Ibaraki, Japan, 1986

39. McGhee, Robert B., "Visual Steering of a Robot Vehicle," *Lecture notes for CS4310*, Naval Postgraduate School, Monterey, California, November 1986.

40. Brainin, S.M. and McGhee, R.B., "Optimal Biased Proportional Navigation," *IEEE Trans. on Automatic Control*, v. AC-13, n. 4, pp. 440-442, August 1968.

41. *IRIS User's Guide*, Silicon Graphics, Inc., Mountain View, California, 1986

42. *UNIX Programmer's Manual Volume IA: Commands*, Silicon Graphics, Inc., Mountain View, California, 1986

43. *UNIX Programmer's Manual Volume IB: System Calls and Subroutines*, Silicon Graphics, Inc., Mountain View, California, 1986

44. *UNIX Programmer's Manual Volume IIB: Languages and Tools*, Silicon Graphics, Inc., Mountain View, California, 1986

45. Fossum, Timothy V. and Gatterdam, Ronald W., *Calculus and the Computer: An Approach to Problem Solving*, Scott Foresman, Glenview, Illinois, 1980.

46. Firth, S. and Zyda, Michael J., *The OZDRAW User's Manual*, Naval Postgraduate School, Graphics and Video Laboratory, Monterey, California, 1985.

APPENDIX - SOURCE PROGRAMS

A. CARSIMU.C

```
/*

This is the main program of the entire vehicle simulation
program.  To recompile this program just issue the command
Makefile.

*/

#include "road.h"

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

   G L O B A L     D E C L A R A T I O N S

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

/* DO NOT remov any of these declarations.
   They may be used in the supporting programs. */

Tag transl4, transl3, transl2, transl1, transl, trans22;
Tag house1looktag, house1transtag, house1scaletag;
Tag houselooktag, housetranstag, housescaletag;
Tag dangertag, temptag, belttag, braketag;
Tag odotag1, odotag2, odotag3, odotag4;
Tag fuel1, roadlooktag, skylooktag;
Tag steerwheeltag, terrain1looktag;

Coord latri[3][2], ratri[3][2];

float fuelbar,speedbar;
float fuelquant    = MAXFUEL;  /* Maximum fuel available    */
float heading_xpos = 429.5;    /* Heading indicator position */
float speedinc     = 1.0;      /* Speed increment/decrement  */

Device keypressed;

Boolean start      = TRUE;     /* Start of program flag      */

/*

  Larger turning_response_gain corresponds to "stiff"
  steering and lower value corresponds to "sloppy"
  steering.  Large velocity_gain corresponds to sedan
  automobile and smaller value corresponds to sport car.
  Operator has control over steer_wheel_angle and speed
  using the mouse.

  Car_time is the integration timer.

*/

float state_vector[5];
float cmd_psi_dot;

float heading_angle_rate_gain = 1.0;
float velocity_time_consant   = 9.0;
float turning_response_gain   = 0.02;
float heading_angle_gain      = 1.0;
float steer_wheel_angle       = 0.0;      /* Unit is radian */
float prediction_time         = 1.0;  /* Unit is second */
float steer_inc               = 0.10;     /* Unit is radian */
float car_time                = 0.0;
float deltat                  = 0.17;
float speed                   = 0.0;

/* IRIS allow such a large array only if it is global */
float roadmap[5000][3];

Dimension Bendradius1 = 80.0;
Dimension Roadwidth   = 16.0;
Dimension Roadlen     = 400.0;
Angle     Fov         = 1000;    /* Field of view 100 deg */

main()
{

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

        L O C A L     D E C L A R A T I O N S

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

int old_sampling_cycle        = -1;
int sampling_interval         = 1;      /* Steering point sampling rate */
int prev_mousex               = 250;    /* Previous mouse x position */
int where                     = 1;      /* Steering point location */
int distance                  = 0;      /* Distance travelled */


char thousandc[2], hundredc[2], tenc[2], unitc[2], temp_string[15];
int i, no_coord, new_sampling_cycle, mousex, cal_mousex;
int count, unit, ten, hundred, thousand, no_of_round;
FILE *fp;

float sigma_dot               = 0.0;
float tolerance               = 1.0;
float old_sigma              = 0;
float sigma                  = 0.0;

float prediction_distance;
float temp, templ;
float gx, gy, gz;


extern long time();  /*  System clock  */
char timec[10];      /*  Car time in char format  */
long clocktime;      /*  For clock value */
char *clockc;

Boolean  showclock  = TRUE;  /* Display clock flag */
Boolean  showtimer  = FALSE; /* Display integration timer */
Boolean  alarm      = FALSE; /* Off road warning flag */
Boolean  bell       = FALSE; /* Turn off danger alarm  */
Boolean  debug      = TRUE;  /* Turn off debug info */
Boolean  autop      = FALSE; /* Turn off debug info */
Boolean  ebrake     = FALSE; /* Emergence brake flag */

Dimension consumption = 1.0;     /* Fuel consumption */
Dimension crashdown   = 0.0;     /* Off-road display flag */
Dimension fueldown    = 0.0; /* Fuel depleted display flag */
Dimension headingdeg  = 0.0;     /* Heading in degrees */
Dimension headingrad  = 0.0;     /* Heading in radians */
Dimension rdistance   = 0.0;   /* Distance travelled */
Dimension vd          = 100.0; /* Viewing distance   */

Dimension mps_to_kmph = 3.6;         /* m/s to km/hr conversion */
Dimension rad_to_deg  = 360/(2*PI);        /* radian to degree conversion */

Coord crashx = 512.0;  /* X viewport coord to detect off-road */
Coord crashy = 385.0;  /* Y viewport coord to detect off-road */

Coord warnx1 = 212.0;  /* X viewport coord to warn off-road */
Coord warnx2 = 812.0;  /* X viewport coord to warn off-road */
Coord warny  = 385.0;  /* Y viewport coord to warn off-road */

Colorindex colors[1];  /* Array to store color of crash spot */
short nopixel = 1;     /* No of pixel to detect off-road */

Coord cx, cy, cz;    /* Current viewing point */
Coord rx, ry, rz;    /* Reference point      */
Coord pz, px;        /* Last viewing point    */

Object speedometer, fuel, steerwheel, signboard, sky, mountain;
Object terrain1, odometer, warning, heading_meter;
Object road, help, arrow, house, house1;

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

```
    SYSTEM    INITIALIZATIONS

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

/* Initial state vector of the automobile */

state_vector[1] = 0.0; /* initial z coord */
state_vector[2] = 0.0; /* initial x coord */
state_vector[3] = 0.0; /* initial velocity */
state_vector[4] = 0.0; /* initial heading */

cx = 0.0; cy = 3.0; cz = 0.0;
rx = 0.0; ry = 3.0; rz = -vd;
count = unit = ten = hundred = thousand = 0;

ginit();
doublebuffer();
gconfig();
cursoff();
qdevice(KEYBD);
viewport(0, XMAXSCREEN, 0, YMAXSCREEN);
ortho2(0.0, 1023.0, 0.0, 767.0);

blink(10, CYAN, 255, 0. 0);
bbox2i(5, 5, 0, 1023, 0, 767);

mapcolor(MOUNTAIN, 199, 123, 63);
mapcolor(MOUNTAIN1, 210, 150, 0);
mapcolor(FIELD, 5, 190, 20);
mapcolor(SKY, 50, 8, 155);
mapcolor(WARN, 125, 0, 0);

mapcolor(CHMWALL1,118,76,0);
mapcolor(CHMWALL2,146,114,0);
mapcolor(WINDOW,0,141,205);
mapcolor(SIDEROOF,188,50,14);
mapcolor(FRAME,118,50,14);
mapcolor(WALL,164,111,0);
mapcolor(SIDEWALL,146,94,1);
mapcolor(ROOF,148,50,14);

/*  Dark Grey  */
mapcolor(ROOF1,100,100,100);
/*  Light Grey  */
mapcolor(FRAME1,0,60,60);
/*  Light Grey  */
mapcolor(SIDEWALL1,150,60,60);
/*  Pink  */
mapcolor(WALL1,160,60,60);

setvaluator(MOUSEX, 250, 0, 500);
setvaluator(MOUSEY, 250, 0, 500);
```

```
noise(MOUSEX, 10);

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

    M A K E   A L L   T H E   O B J E C T S

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

makethespeedometer(&speedometer);
makeheading(&heading_meter);
makesteerwheel(&steerwheel);
maketheodometer(&odometer);
maketerrain1(&terrain1);
makewarning(&warning);
maketheroad(&road);
makehouse1(&house1);
makehouse(&house);
makethesky(&sky);
makehelp(&help);
makefuel(&fuel);

/* Display the introductory image */
welcome();

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

        R E A D   R O A D M A P

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

/* Read road map into system */
if ((fp = fopen("roadmap","r")) == NULL)
    {
    printf("Cannot read roadmap.\n");
    return(-1);
    }
else
    for (i = 0; !feof(fp); ++i)
        fscanf(fp,"%f %f %f", &roadmap[i][0], &roadmap[i][1],
            &roadmap[i][2]);
no_coord = --i;


setbell('1');
ringbell();
setbell('2');
ringbell();

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

    I N I T I A L I Z E   B U F F E R S
```

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

/* Wait till a mouse is pressed. */
while(getbutton(MOUSE3) == 0);
color(BLACK);
clear();
swapbuffers();
clear();
swapbuffers();

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

    M A I N   S I M U L A T I O N   L O O P

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

while(TRUE)
    {
    new_sampling_cycle = count/sampling_interval;

    pz = cz;
    px = cx;

    clocktime = time((long *) 0);
     clockc = ctime(&clocktime);

    /* To display clock? */

    if (keypressed == 'c' || keypressed == 'C')
        if (showclock) showclock = FALSE;
            else showclock = TRUE;

    /*  Sound alarm around 2m before off the road */

    cmov2(warnx1, warny);
    readpixels(nopixel, colors);
    if (colors[0] != BLACK && colors[0] != WHITE)
        alarm = TRUE;
        else alarm = FALSE;

    if (!alarm)
        {
        cmov2(warnx2, warny);
        readpixels(nopixel, colors);
        if (colors[0] != BLACK && colors[0] != WHITE)
            alarm = TRUE;
            else alarm = FALSE;
        }

    /* Check if the vehicle is off the road
      IMPT : Assume road surface is black
          and surface signs are white    */
    cmov2(crashx, crashy);
    readpixels(nopixel, colors);
    if (colors[0] != BLACK && colors[0] != WHITE)
        crashdown = -1000.0;

    rz = - (vd*cos(state_vector[4]) + state_vector[1]);
    rx = vd*sin(state_vector[4]) + state_vector[2];

    if (keypressed == 'q' || keypressed == 'Q')
        if (autop)
            {
            autop = FALSE;
            prev_mousex = mousex;
            }
        else if (state_vector[3] > 0) autop = TRUE;

#ifdef DEBUG
    for(i = 1; i <= SYSTEM_ORDER; ++i)
        switch(i)
        {
        case 1:
            printf("X: %.2f  ",state_vector[1]);
            break;
        case 2:
            printf("Y: %.2f  ",state_vector[2]);
            break;
        case 3:
            printf("Velocity: %.2f  ",state_vector[3]*mps_to_kmph);
            break;
        case 4:
            printf("Heading: %.2f\n",state_vector[4] * rad_to_deg);
            break;
        }
#endif

    cz = -state_vector[1];
    cx = state_vector[2];

    /*  Check if keyboard pressed.  Keys pressed are queued. */

    checkkeybd();

    mousex = getvaluator(MOUSEX);

    if (!autop && !ebrake)
        {
        if (getbutton(MOUSE1) && getbutton(MOUSE2)
            && getbutton(MOUSE3))

            /*  Exit Program  */

            break;
        else if (getbutton(MOUSE1) || (keypressed == 'a' ||
            keypressed == 'A'))
        {
        if (speed < 190)
            {
            start = FALSE;
            speed = speed + speedinc;
            }
        else  speed = 190.0;  /*  Top Speed */
        }

        else if (getbutton(MOUSE3) || (keypressed == 'e' ||
            keypressed == 'E'))
        {

        /*  Emergency brake  */
        ebrake = TRUE;
        /* state_vector[3] = 0.0;
        speed = 0.0;  */
        }

        else if (getbutton(MOUSE2) || (keypressed == 'b' ||
            keypressed == 'B'))
        {

        /*  Decrease speed  */

        if (speed > 0)
            speed = speed - speedinc;
        else  speed = 0.0;
        }

    }  /* if (!autop) */

if (!ebrake && state_vector[3] > 0)
    {
    prediction_distance = state_vector[3] * prediction_time;

    /*
    "where" is passed to find_subgoal so that searching
    need not always start from the beginning of the road.
    The z and y convention in the graphics system is reversed.
    Also the sign is going in the opposite direction.  So
    compensate before passing into find_subgoal.
    */

    where = find_subgoal(roadmap, no_coord, where, tolerance,
            prediction_distance, cx, -cz, 0.0);
    }

if (!ebrake && (autop && old_sampling_cycle < new_sampling_cycle
    || !autop))
```

```
        {
        old_sampling_cycle = new_sampling_cycle;
        if (where < 0)
            {
            /* Stop completely and remove autopilot */
            ebrake = TRUE;
            /* state_vector[3] = 0.0;
            speed = 0.0;  */
            autop = FALSE;
            }
        else
            {
            gx = roadmap[where][0];
            gy = roadmap[where][1];
            gz = roadmap[where][2];
            }
        }

    /* Convention difference: Z-axis in graphics is Y-axis in
    mathematical model.  Also Z-axis is negative when moving
    into the screen which therefore must be converted to
    positive for our calculation. */

    temp = -cz;
    sigma = atan2((gx-cx),(gy-temp));

    /* Sigma_dot(0) = 0 */
    if (count == 0) old_sigma = sigma;

    /* This is 45 deg branch cut to handle the discontinuity
    when arc tangent function crosses PI and -PI */

    if (sigma < -(PI/4)) sigma = 2*PI + sigma;

#ifdef DEBUG
        printf("gx %.2f cx %.2f gy %.2f cz %.2f",gx, cx, gy, cz);
        printf("  sigma_deg %.2f\n",(sigma/(2*PI))*360);
#endif

    sigma_dot = (sigma - old_sigma)/deltat;

    cmd_psi_dot  = (heading_angle_rate_gain * sigma_dot) +
            (heading_angle_gain * (sigma - state_vector[4]));

    if (autop)
        steer_wheel_angle = cmd_psi_dot/
            (turning_response_gain * state_vector[3]);

    old_sigma = sigma;

    if (!ebrake && !autop)
        {
```

```
        /* Manual Driving */

        cal_mousex = mousex - prev_mousex;
        steer_wheel_angle = steer_wheel_angle + (float) cal_mousex/100;
        prev_mousex = mousex;
        }

compute_new_state(autop);

/*  Clear the vehicle window  */

viewport(0, XMAXSCREEN, 385, YMAXSCREEN);
color(FIELD);
clear();

/*  Clear the display panel  */

viewport(0, XMAXSCREEN, 0, 380);
color(WHITE);
clear();

/*  Reset viewport  */

viewport(0, XMAXSCREEN, 0, YMAXSCREEN);

/* Calculate the velocity for emergence brake */
if (ebrake)
        {
        /* every 16.0 km/hr or 4.0 mph will take the
        vehicle one additional cycle to stop. */

        state_vector[3] = state_vector[3] - 4.0;
        speed = state_vector[3];
        if (state_vector[3] < 0)
                {
                ebrake = FALSE;
                state_vector[3] = 0.0;
                speed = 0.0;
                }
        }

/*  Calculate distance travelled  */

rdistance = rdistance+sqrt((cz-pz)*(cz-pz)+(cx-px)*(cx-px));

if (keypressed == 'o' || keypressed == 'O') rdistance = 0.0;

distance = (int) rdistance;
thousand = distance/1000;
hundred = (distance - thousand*1000)/100;
ten = (distance - hundred * 100 - thousand*1000)/10;
unit = distance - ten * 10  - hundred * 100 - thousand*1000;
```

```
if (unit == 10) { unit = 0; ++ten; }
if (ten == 10) { ten = 0; ++hundred; }
if (hundred == 10) { hundred = 0; ++thousand; }
if (thousand == 10) thousand = 0;
sprintf(timec,"%5.2f",car_time);
sprintf(thousandc,"%d",thousand);
sprintf(hundredc,"%d",hundred);
sprintf(tenc,"%d",ten);
sprintf(unitc,"%d",unit++);

/* DISPLAY HELP PANEL  */

callobj(help);

/*  EDIT SKY  */

editobj(sky);
objreplace(skylooktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
closeobj();
callobj(sky);

/*  EDIT TERRAIN  */

editobj(terrain1);
objreplace(terrain1looktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
closeobj();
callobj(terrain1);

/*  EDIT ROAD  */

editobj(road);
objreplace(roadlooktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
closeobj();
callobj(road);

/*  EDIT HOUSES  */

editobj(house);
objreplace(houselooktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
objreplace(housetranstag);
translate(-80.0, 0.0, -50.0);
objreplace(housescaletag);
scale(0.40, 0.40, 1.0);
closeobj();
callobj(house);

editobj(house1);
objreplace(house1looktag);
```

```
lookat(cx,cy,cz,rx,ry,rz,0.0);
objreplace(house1transtag);
translate(-30.0, 0.0, -10.0);
objreplace(house1scaletag);
scale(0.50, 0.50, 1.0);
closeobj();
callobj(house1);

editobj(house1);
objreplace(house1looktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
objreplace(house1transtag);
translate(-30.0, 0.0, -15.0);
objreplace(house1scaletag);
scale(0.50, 0.50, 1.0);
closeobj();
callobj(house1);

editobj(house);
objreplace(houselooktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
objreplace(housetranstag);
translate(100.0, 0.0, -Roadlen/2);
objreplace(housescaletag);
scale(0.50, 0.50, 1.0);
closeobj();
callobj(house);

editobj(house);
objreplace(houselooktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
objreplace(housetranstag);
translate(-40.0, 0.0, -Roadlen - 100.0);
objreplace(housescaletag);
scale(0.80, 0.80, 1.0);
closeobj();
callobj(house);

editobj(house1);
objreplace(house1looktag);
lookat(cx,cy,cz,rx,ry,rz,0.0);
objreplace(house1transtag);
translate(300.0, 0.0, -Roadlen - 55.0);
objreplace(house1scaletag);
scale(0.50, 0.50, 1.0);
closeobj();
callobj(house1);

/* EDIT STEERING WHEEL */

editobj(steerwheel);
objreplace(steerwheeltag);
```

```
rotate((int) -(steer_wheel_angle * 10 * rad_to_deg), 'Z');
closeobj();
callobj(steerwheel);

/*  EDIT ODOMETER  */

editobj(odometer);
objreplace(odotag1);
charstr(thousandc);
objreplace(odotag2);
charstr(hundredc);
objreplace(odotag3);
charstr(tenc);
objreplace(odotag4);
charstr(unitc);
closeobj();
callobj(odometer);

if (showclock)
      {
      color(WHITE);
      cmov2i(100, 750);
      charstr(clockc);
      color(BLACK);
      }

if (autop)
      {
      color(CYAN);
      cmov2i(400, 750);
      charstr("AutoPilot Mode");
      color(BLACK);
      }


/*  TESTING AREA */

if (keypressed == 'z' || keypressed == 'Z')
      if (debug) debug = FALSE;
            else debug = TRUE;

if (debug)
      {
      cmov2i(575, 280);
      charstr("Command Speed (km/h) ");
      sprintf(temp_string,"%.2f",speed * mps_to_kmph);
      charstr(temp_string);

      cmov2i(575, 250);
      charstr("Steering (degree) ");
      sprintf(temp_string,"%.2f",steer_wheel_angle * rad_to_deg);
      charstr(temp_string);
```

```
    /* cmov2i(575, 220);
    charstr("Mousex ");
    sprintf(temp_string,"%d",cal_mousex);
    charstr(temp_string);  */

    cmov2i(575, 180);
    charstr("Turning Response Gain ");
    sprintf(temp_string,"%.3f",turning_response_gain);
    charstr(temp_string);

    cmov2i(575, 150);
    charstr("Heading Angle Gain ");
    sprintf(temp_string,"%.3f",heading_angle_gain);
    charstr(temp_string);

    cmov2i(575, 120);
    charstr("Prediction Time ");
    sprintf(temp_string,"%.2f",prediction_time);
    charstr(temp_string);
    }

/*  EDIT TIMER  */

if (keypressed == 't' || keypressed == 'T')
    if (showtimer) showtimer = FALSE;
        else showtimer = TRUE;

if (showtimer)
    { cmov2i(575, 310);
    charstr("Integration: ");
    charstr(timec); }

/*  EDIT WARNING INDICATOR  */

if (state_vector[3] > 0)
    { editobj(warning);
    objreplace(braketag);
    color(WARN);
    if (alarm)
        {
        if (keypressed == 's' || keypressed == 'S')
            if (bell) bell = FALSE;
                else bell = TRUE;
        if (bell)
            {
            setbell('2');
            ringbell();
            }
        objreplace(dangertag);
        color(CYAN);
        }
    else
```

```
            {
            objreplace(dangertag);
            color(WARN);
            }

      /*  ENGINE WARMING UP  */

      if (count < 1000)
            { objreplace(temptag);
            color(WARN); }

      /*  ENGINE REACHED NORMAL TEMPERATURE  */

      if ((count > 1000) && (count < 5000))
            { objreplace(temptag);
            color(YELLOW); }

      /*  ENGINE OVERHEATING  */

      if (count > 5000)
            { objreplace(temptag);
            color(RED); }

      objreplace(belttag);
      color(WARN);
      closeobj(); }
  else

      /*  BRAKE SIGNAL FOR CAR STOP  */

      {
      editobj(warning);
      objreplace(braketag);
      color(RED);
      closeobj();
      }

callobj(warning);

/*  EDIT HEADING INDICATOR  */

/*  Compute heading using vehicle state vector  */

if (state_vector[4] < 0.0) headingrad = 2*PI+state_vector[4];
      else headingrad = state_vector[4];
no_of_round = (headingrad*180.0/PI)/360.0;
headingdeg = headingrad*180.0/PI - (float) no_of_round*360;

editobj(heading_meter);
objreplace(transl1);
translate(heading_xpos-20.0-4.5*headingdeg, 4.0, 0.0);
closeobj();
```

```
callobj(heading_meter);

/*  EDIT SPEEDOMETER INDICATOR  */

/*  2.5 factor is for converting to the dashboard display */

speedbar = 181.0 - state_vector[3] * mps_to_kmph * 2.5;

 editobj(speedometer);
objreplace(transl4);
translate(0.0, speedbar, 0.0);
closeobj();
callobj(speedometer);

/*  EDIT FUEL GAUGES  */

/* Stop : no consumption */
/* Speed above 100 : consumption is 20% higher */

if (state_vector[3] > 0.0)
      if (state_vector[3] < 100.0)
            fuelquant = fuelquant - consumption;
            else fuelquant = fuelquant - 1.2*consumption;

fuelbar = fuelquant/MAXFUEL*320.0+14.0;

if (fuelquant < 0.0) fueldown = -1000.0;

editobj(fuel);
objreplace(fuel1);
rectf(106.0,14.0,149.0,fuelbar);
closeobj();
callobj(fuel);

 /* EDIT CRASH INFO DISPLAY FOR OFF-ROAD */

 pushmatrix();
 pushattributes();

 translate(0.0,crashdown,0.0);

/*  Set all warning lights when crash  */

if (crashdown == -1000)
      {
      autop = FALSE;

      editobj(warning);

      objreplace(braketag);
      color(RED);
```

```
     bell = FALSE;
     objreplace(dangertag);
     color(RED);

     objreplace(temptag);
     color(RED);

     objreplace(belttag);
     color(RED);

     closeobj();

     callobj(warning);

     }

 color(RED);
 rectf(0.0,1385.0,1023.0,1767.0);

color(BLACK);
cmov2i(370,1576);
charstr("CRASH");
 cmov2i(370,1560);
charstr("OFF THE ROAD");

 cmov2i(370,1544);
charstr("PUSH ALL THREE MOUSE BUTTONS TO EXIT");

 popattributes();
 popmatrix();

 /* EDIT CRASH INFO DISPLAY FOR FUEL DELETION */

 pushmatrix();
 pushattributes();

 translate(0.0,fueldown,0.0);

 color(MAGENTA);
 rectf(0.0,1385.0,1023.0,1767.0);


color(BLACK);
cmov2i(370,1576);
charstr("STOP");
 cmov2i(370,1560);
charstr("FUEL DEPLETED");

 cmov2i(370,1544);
charstr("PUSH ALL THREE MOUSE BUTTONS TO EXIT");

 popattributes();
```

```
         popmatrix();


        swapbuffers();
        ++count;
        } /* while loop */

color(BLACK);
clear();
swapbuffers();
clear();
swapbuffers();
finish();
gexit();

} /* main */

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

          C H E C K    K E Y B O A R D

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

/* Keyboard keys can be used to controlled steer_wheel_angle.
   Keys pressed are queued whereas the mouse is not.
   Keys increase the angle by a smaller amount.    */

checkkeybd()
{

keypressed = NULL;

if (qtest())
        {
        qread(&keypressed);

        /*  Display help information  */

        if (keypressed == 'h' || keypressed == 'H')
                help();

        /* printf("%d\n",keypressed); */
        }
}  /* checkkeybd */
```

B. CIRCUIT.C

```
/*

Build the entire road circuit.

*/

#include "road.h"

extern Tag roadlooktag:
extern float Roadwidth, Roadlen;
extern float Bendradius1:
extern Angle Fov:

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

   B U I L D    T H E    R A L L Y    C I R C U I T

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

maketheroad(road)
Object *road;
{
Dimension  temp, i:
Dimension  high      = 3.2;
Colorindex signbg    = YELLOW;
Colorindex upsign    = RED;
Colorindex rightsign = BLUE:

*road = genobj():
makeobj(*road);
pushmatrix();
pushviewport():
viewport(0, XMAXSCREEN, 385, YMAXSCREEN);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0. 0.0, 1023.0);
roadlooktag = gentag();
maketag(roadlooktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0);

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

   FIRST STRETCH OF ROAD

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

surf(0.0, 0.0, 0.0, Roadwidth, Roadlen, BLACK);

/*
```

```
Build the sign "START"

*/

temp = -5.5;
color(WHITE);
pushmatrix();
translate(temp + 4.0, 0.0, 0.0);
rotate(-900,'X');
letter('T', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp + 2.0, 0.0, 0.0);
rotate(-900,'X');
letter('R', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp, 0.0, 0.0);
rotate(-900,'X');
letter('A', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp - 2.0, 0.0, 0.0);
rotate(-900,'X');
letter('T', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp - 4.0, 0.0, 0.0);
rotate(-900,'X');
letter('S', BLACK);
popmatrix();

/*

Build the sign "TURN" before the bend

*/

color(WHITE);
pushmatrix();
translate(-2.0, 0.0, -(Roadlen - 5.0));
rotate(-900,'X');
letter('N', BLACK);
popmatrix();
```

```
color(WHITE);
pushmatrix();
translate(-4.0, 0.0, -(Roadlen - 5.0));
rotate(-900,'X');
letter('R', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(-6.0, 0.0, -(Roadlen - 5.0));
rotate(-900,'X');
letter('U', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(-8.0, 0.0, -(Roadlen - 5.0));
rotate(-900,'X');
letter('T', BLACK);
popmatrix();


/*

Build a series of arrow

*/

for (temp = 8.0; temp < Roadlen; temp += 40.0)
      {
      pushmatrix();
      translate(0.0, 0.0, -temp);
      rotate(-900,'X');
      polyarrow(0.7, 1.2, 0.0, WHITE);
      popmatrix();
      }

/*

Create 1st uparrow signboard

*/

pushmatrix();
translate(10.0, 0.0, -5.0);
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(10.0, high, -5.0);
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();
```

```
/*

Create 2nd uparrow signboard

*/

pushmatrix();
translate(-7.0, 0.0, -(Roadlen/3.0));
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(-7.0, high, -(Roadlen/3.0));
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();

/*

First road bend

*/

pushmatrix();
translate(Bendradius1 - Roadwidth/2, 0.0, -Roadlen);
rotate(-900, 'X');
bend();
popmatrix();

/*

Build 1nd right turn signboard

*/

pushmatrix();
translate(7.0, 0.0, -(Roadlen-5.0));
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(6.3, 4.0, -(Roadlen-5.0));
rotate(-900,'Z');
polyarrow(0.7, 1.2, 0.0, rightsign);
popmatrix();

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

   SECOND STRETCH OF ROAD

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

pushmatrix();
temp = Bendradius1 - Roadwidth/2;
```

```
translate(temp, 0.0, -Roadlen - temp);
rotate(-900, 'Y');
surf(0.0, 0.0, 0.0, Roadwidth, Roadlen, BLACK);
popmatrix();

/*

Build a series of road strips

*/

for (i = temp + 8.0; i < Roadlen; i += 20.0)
      {
      pushmatrix();
      translate(i, 0.0, -Roadlen - temp);
      surf(0.0, 0.0, 0.0, 3.0, 1.0, WHITE);
      popmatrix();
      }

/*

Create 3rd uparrow signboard

*/

pushmatrix();
translate(temp + 50.0, 0.0, -Roadlen - temp - Roadwidth);
rotate(-900,'Y');
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(temp + 50.0, high, -Roadlen - temp - Roadwidth);
rotate(-900,'Y');
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();

/*

Second road bend

*/

pushmatrix();
temp = Bendradius1 - Roadwidth/2 + Roadlen;
translate(temp, 0.0, -Roadlen);
rotate(-900, 'X');
rotate(-900, 'Z');
bend();
popmatrix();

/*
```

```
Build 2nd right turn signboard

*/

pushmatrix();
translate(temp - Roadwidth, 0.0, -Roadlen - Bendradius1);
rotate(-900,'Y');
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(temp - Roadwidth, 4.0, -Roadlen - Bendradius1 - 0.7);
rotate(-900,'Y');
rotate(-900,'Z');
polyarrow(0.7, 1.2, 0.0, rightsign);
popmatrix();


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

   THIRD STRETCH OF ROAD

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


pushmatrix();
temp = 2 * Bendradius1 - Roadwidth + Roadlen;
translate(temp, 0.0, 0.0);
surf(0.0, 0.0, 0.0, Roadwidth, Roadlen, BLACK);
popmatrix();

/*

Create a series of arrows

*/

for (i = Roadlen; i > 5.0; i -= 20.0)
      {
      pushmatrix();
      translate(temp, 0.0, -i);
      rotate(-900,'X');
      rotate(-1800, 'Z');
      polyarrow(0.7, 1.2, 0.0, WHITE);
      popmatrix();
      }

/*

Create 4nd uparrow signboard

*/
```

```
pushmatrix();
translate(temp + Roadwidth, 0.0, -Roadlen + 10.0);
rotate(-1800,'Y');
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(temp + Roadwidth, high, -Roadlen + 10.0);
rotate(-1800,'Y');
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();

/*

Third road bend

*/

pushmatrix();
temp = Bendradius1 - Roadwidth/2 + Roadlen;
translate(temp, 0.0, 0.0);
rotate(-900, 'X');
rotate(-1800, 'Z');
bend();
popmatrix();


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

   FOURTH STRETCH OF ROAD

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

pushmatrix();
temp = Bendradius1 - Roadwidth/2;
translate(temp, 0.0, temp);
rotate(-900, 'Y');
surf(0.0, 0.0, 0.0, Roadwidth, Roadlen, BLACK);
popmatrix();

/*

Create a series of arrows

*/

for (i = temp + 10.0; i < temp + Roadlen; i += 20.0)
      {
      pushmatrix();
      translate(i, 0.0, temp);
      rotate(-900,'X');
      rotate(900,'Z');
      polyarrow(0.7, 1.2, 0.0, WHITE);
```

```
    popmatrix();
    }

/*

Create 5nd uparrow signboard

*/

pushmatrix();
translate(Roadlen, 0.0, Bendradius1 + 2.0);
rotate(-2700,'Y');
signb(1.9, 2.5, 3.0, signbg);
popmatrix();
pushmatrix();
translate(Roadlen, high, Bendradius1 + 2.0);
rotate(-2700,'Y');
polyarrow(0.7, 1.2, 0.0, upsign);
popmatrix();

/*

"STOP" sign before the end of circuit

*/

temp = 1.5 * Bendradius1;
color(WHITE);
pushmatrix();
translate(temp, 0.0, 70.0);
rotate(900,'Y');
rotate(-900,'X');
letter('P', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp, 0.0, 75.0);
rotate(900,'Y');
rotate(-900,'X');
letter('O', BLACK);
popmatrix();

color(WHITE);
pushmatrix();
translate(temp, 0.0, 80.0);
rotate(900,'Y');
rotate(-900,'X');
letter('T', BLACK);
popmatrix();

color(WHITE);
```

```
pushmatrix();
translate(temp, 0.0, 84.0);
rotate(900,'Y');
rotate(-900,'X');
letter('S', BLACK);
popmatrix();

popviewport();
popmatrix();
closeobj();
}  /* maketheroad */
```

C. INTEGRATE.C

```
/*

Runge-Kutta 2nd order or Euler-Heun numerical integration

*/

#include "road.h"

extern float heading_angle_rate_gain, velocity_time_consant;
extern float turning_response_gain, heading_angle_gain;
extern float steer_wheel_angle, speed, state_vector[5];
extern float car_time, deltat, cmd_psi_dot;

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

          I N T E G R A T I O N

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

compute_new_state(autop)
Boolean autop;

{
float xcap[5], xdot[5];
int i;

derivative(state_vector, xdot, autop);
for (i = 1; i <= SYSTEM_ORDER; ++i)
     /*  Euler prediction  */
     xcap[i] = state_vector[i] + xdot[i] * deltat;

car_time = car_time + deltat;
derivative(xcap, xdot, autop);
for (i = 1; i <= SYSTEM_ORDER; ++i)
     /* Trapezodial correction */
     state_vector[i] = (state_vector[i] + xdot[i]
                  * deltat + xcap[i])/2.0;

}  /* compute_new_state */

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

          D E R I V A T I V E

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

derivative(work_vector, xdot, autop)
Boolean autop;
float work_vector[], xdot[];
```

```
{

xdot[1] = cos(work_vector[4]) * work_vector[3];
xdot[2] = sin(work_vector[4]) * work_vector[3];

if (!autop)
      xdot[3] = - (1/velocity_time_consant) * work_vector[3]
                  + (1/velocity_time_consant) * speed;
else
      xdot[3] = 0;  /* no acceleration for autopilot */

if (!autop)
      xdot[4] = turning_response_gain * work_vector[3]
            * steer_wheel_angle;
else
      {
      xdot[4] = cmd_psi_dot;
      }

}  /* derivative */
```

D. DISPLAY.C

```
/*

This module generates all the vehicle dashboard indicators.

        1. Speedometer
        2. Odometer
        3. Compass
        4. Fuel Meter
        5. Help Panel
        6. Warning Panel
        7. Steering Wheel Display

Some of the ideas here were adopted from fltsim.c.

*/

#include "road.h"

extern Coord latri[3][2], ratri[3][2];
extern Tag transl, transl2, transl3, transl4, trans22, fuel1;
extern Tag odotag1, odotag2, odotag3, odotag4, steerwheeltag;
extern Tag dangertag, temptag, belttag, braketag, transl1;
extern float heading_xpos;


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

          S P E E D O M E T E R

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


makethespeedometer(speedometer)
Object *speedometer;

{
Icoord charxpos, pos1, pos2, tempx, tempy;
Object meter,meternum;

pos1 = 467; pos2 = 150;
tempx = pos1 + 90;
tempy = pos2 + 80;
charxpos = pos1 + 30;

/*  Generate outline for speedometer dial  */

meter=genobj();

makeobj(meter);
```

```
color(BLACK);

rectfi(pos1, pos2, tempx, tempy);

color(WHITE);
rectfi(pos1+10, pos2+10, tempx-10, tempy-10);
color(BLACK);


cmov2i(pos1,pos2-15);
charstr(" km/hr ");

latri[0][0]=pos1;
latri[0][1]=190-9;

latri[1][0]=pos1+25;
latri[1][1]=190;

latri[2][0]=pos1;
latri[2][1]=190+9;

polf2(3,latri);

ratri[0][0]=tempx;
ratri[0][1]=190-9;

ratri[1][0]=tempx;
ratri[1][1]=190+9;

ratri[2][0]=tempx-25;
ratri[2][1]=190;
polf2(3,ratri);

closeobj();

/*  Generate number in speedometer display  */

meternum=genobj();

makeobj(meternum);

color(BLACK);

cmov2i(charxpos,000);
charstr("000");
cmov2i(charxpos,030);
charstr("010");
cmov2i(charxpos,060);
charstr("020");
cmov2i(charxpos,090);
charstr("030");
cmov2i(charxpos,100);
```

```
charstr("040");
cmov2i(charxpos,125);
charstr("050");
cmov2i(charxpos,150);
charstr("060");
cmov2i(charxpos,175);
charstr("070");
cmov2i(charxpos,200);
charstr("080");
cmov2i(charxpos,225);
charstr("090");
cmov2i(charxpos,250);
charstr("100");
cmov2i(charxpos,275);
charstr("110");
cmov2i(charxpos,300);
charstr("120");
cmov2i(charxpos,325);
charstr("130");
cmov2i(charxpos,350);
charstr("140");
cmov2i(charxpos,375);
charstr("150");
cmov2i(charxpos,400);
charstr("160");
cmov2i(charxpos.425);
charstr("170");
cmov2i(charxpos,450);
charstr("180");
cmov2i(charxpos,475);
charstr("190");

closeobj();

/*   Put all pieces of speedometer  together   */


*speedometer=genobj();

makeobj(*speedometer);


/* Draw the boundary  */

callobj(meter);

/*  Draw the display speedometer in the  window */

scrmask(charxpos,tempx,pos2+10,tempy-10);

pushmatrix();
transl4=gentag();
```

```
maketag(transl4);
translate(0.0,0.0,0.0);
callobj(meternum);
popmatrix();

/*  Reset screenmask to full size screen  */
scrmask(0,1023,0,767);

viewport(0,1023,0,767);

closeobj();

}  /*  makethespeedometer  */


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

        F U E L   M E T E R

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


makefuel(fuel)
Object *fuel;

{

Coord fuelx1, fuelx2, fuely1, fuely2;
Object fuelbound,fuellevel;

fuelx1 = 102.0; fuelx2 = fuelx1 + 51.0;
fuely1 = 10.0; fuely2 = 340.0;

/* Generate outline for fuel indicator */

fuelbound=genobj();
makeobj(fuelbound);

color(BLACK);
rectf(fuelx1, fuely1, fuelx2, fuely2);


cmov2(107.0,345.0);
charstr("fuel");

/* Generate hash marks for fuel levels */

linewidth(3);

move(fuelx2, fuely2-30.0, 0.0);
rdr(5.0, 0.0, 0.0);
```

```
/* cmov2(fuelx2+6.0, fuely2-35.0);
charstr(" Full"); */

move(fuelx2, fuely1+60.0, 0.0);
rdr(5.0, 0.0, 0.0);

/* cmov2(fuelx2+6.0, fuely1+55.0);
charstr(" Empty"); */

linewidth(1);

closeobj();

/* Generate the fuel level bar that moves */

fuellevel=genobj();
makeobj(fuellevel);


color(WHITE);
rectf(fuelx1+4.0, fuely1+4.0, fuelx2-4.0, fuely2-4.0);

closeobj();

/* Put all pieces of fuel together */

*fuel=genobj();
makeobj(*fuel);
callobj(fuelbound);

callobj(fuellevel);
color(YELLOW);

fuel1 = gentag();
maketag(fuel1);
rectf(fuelx1+4.0, fuely1+4.0, fuelx2-4.0, fuely2-4.0);
color(BLACK);

closeobj();

} /* makefuel */


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

        H E L P   P A N E L

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


makehelp(help)
Object *help;
```

```
{

*help=genobj();
makeobj(*help);

color(BLACK);


rectfi(10,10,90,340);

color(WHITE);

rectfi(15.15,85,335);

color(BLACK);

/* Generate info on display */

linewidth(2);

cmov2i(10,345);
charstr("   help   ");

cmov2i(32,315);
charstr("Exit");
color(RED);
circfi(29,300,6);
circfi(50,300,6);
circfi(71,300,6);

color(BLACK);
cmov2i(21,275);
charstr(" Speed ");
circi(29,260,6);
circi(50,260,6);
color(RED);
circfi(71,260,6);

color(BLACK);
cmov2i(21,235);
charstr(" Brake");
color(BLACK);
circi(29,220,6);
color(RED);
circfi(50,220,6);
color(BLACK);
circi(71,220,6);

cmov2i(21,195);
charstr("  Stop");
color(RED);
circfi(29,180,6);
```

```
color(BLACK);
circi(50,180,6);
circi(71,180,6);

cmov2i(10,115);
charstr(" Press ");
cmov2i(10,95);
charstr("   h   ");
cmov2i(10,75);
charstr("  for  ");
color(RED);
cmov2i(10,55);
charstr(" HELP");

/*
cmov2i(21,135);
charstr(" Brake");
circi(29.120,6);
color(RED);
circfi(50,120,6);
color(BLACK);
circi(71,120,6);

cmov2i(32,95);
charstr("STOP");
color(BLACK);
circi(29,80,6);
color(RED);
circfi(50,80,6);
circfi(71,80,6);
*/

color(BLACK);

closeobj();

} /* makehelp */


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

             ODOMETER

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

maketheodometer(odometer)
Object *odometer;
{
Icoord pos1, pos2, tempx, tempy;
Coord temp, charx, chary;
pos1 = 467; pos2 = 50;
tempx = pos1 + 90; tempy = pos2 + 50;
```

```
*odometer = genobj();
makeobj(*odometer);

color(BLACK);
rectfi(pos1, pos2, tempx, tempy);
color(WHITE);
rectfi(pos1+5, pos2+5, tempx-5, tempy-5);
color(BLACK);

temp = (tempx - pos1 - 10)/4;
move2(pos1+5+temp, pos2+5);
draw2(pos1+5+temp, tempy-5);

move2(pos1+5+temp*2, pos2+5);
draw2(pos1+5+temp*2, tempy-5);

move2(pos1+5+temp*3, pos2+5);
draw2(pos1+5+temp*3, tempy-5);

move2(pos1+5+temp*4, pos2+5);
draw2(pos1+5+temp*4, tempy-5);

charx = pos1+5+temp/2; chary = (tempy-pos2)/2+pos2-5.0;

cmov2(charx, chary);
odotag1 = gentag();
maketag(odotag1);
charstr("0");

cmov2(charx + temp, chary);
odotag2 = gentag();
maketag(odotag2);
charstr("0");

cmov2(charx + temp*2, chary);
odotag3 = gentag();
maketag(odotag3);
charstr("0");

cmov2(charx + temp*3, chary);
odotag4 = gentag();
maketag(odotag4);
charstr("0");

color(BLACK);
cmov2i(pos1,pos2-15);
charstr("   meter");
closeobj();
} /* maketheodometer */

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

```
          WARNING    PANEL

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


makewarning(warning)
Object *warning;
{
Coord tempx, tempy, pos1, pos2;
Coord ix, iy, tempy1, tempy2, tempy3, tempy4, hg;

pos1 = 840.0; pos2 = 10.0;
tempx = pos1 + 140.0; tempy = 340.0;
iy = ix = 20.0;

*warning = genobj();
makeobj(*warning);

color(BLACK);
rectf(pos1, pos2, tempx, tempy);
hg = (330 - 5*iy)/4;

dangertag = gentag();
maketag(dangertag);
color(RED);
rectf(pos1+ix, pos2+iy, tempx-ix, pos2+iy+hg);
color(BLACK);
cmov2(pos1 + (tempx-pos1)/2 - 25.0, pos2+iy+hg/2-5.0);
charstr("Danger");

temptag = gentag();
maketag(temptag);
color(RED);
rectf(pos1+ix,pos2+iy*2+hg,tempx-ix,pos2+iy*2+2*hg);
color(BLACK);
cmov2(pos1 + (tempx-pos1)/2 - 12.0, pos2+iy*2+hg+hg/2-5.0);
charstr("Temp");

belttag = gentag();
maketag(belttag);
color(RED);
rectf(pos1+ix,pos2+iy*3+hg*2,tempx-ix,pos2+iy*3+3*hg);
color(BLACK);
cmov2(pos1 + (tempx-pos1)/2 - 40.0, pos2+iy*3+hg*2+hg/2-5.0);
charstr("Seat Belt");

braketag = gentag();
maketag(braketag);
color(RED);
rectf(pos1+ix,pos2+iy*4+hg*3,tempx-ix,pos2+iy*4+4*hg);
color(BLACK);
cmov2(pos1 + (tempx-pos1)/2 - 17.0, pos2+iy*4+hg*3+hg/2-5.0);
```

```
charstr("Brake");

color(BLACK);
cmov2(pos1+20.0, tempy+5.0);
charstr("  Warning");

closeobj();
}  /* makewarning */


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

        STEERING   WHEEL

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


makesteerwheel(steerwheel)
Object *steerwheel;
{
*steerwheel = genobj();
makeobj(*steerwheel);

pushmatrix();
color(BLACK);
circfi(512, 290, 40);
color(WHITE);
circfi(512, 290, 30);
color(BLACK);

translate(512.0, 290.0, 0.0);
steerwheeltag = gentag();
maketag(steerwheeltag);
rotate(0, 'Z');
rectfi(-33, -5, 33, 5);

popmatrix();
closeobj();

}  /* makesteerwheel */


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

        HEADING   METER

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


makeheading(heading_meter)
Object *heading_meter;
{
```

```
Object meter, theading;
Coord pos1, pos2, tempx, tempy;
pos1 = heading_xpos; pos2 = 350.0;
tempx = pos1 + 175.0;
tempy = pos2 + 12.5;

meter = genobj();
makeobj(meter);
color(BLACK);
rectf(pos1-2.5, pos2-2.5, tempx+2.5, tempy+2.5);
color(WHITE);
rectf(pos1, pos2, tempx, tempy);
closeobj();

/*  Generate the heading on top of the terrain map  */

theading=genobj();
makeobj(theading);
color(BLACK);



cmov2(000.0,pos2-2.0);
charstr("340");

cmov2(045.0,pos2-2.0);
charstr("350");

cmov2(090.0,pos2-2.0);
charstr("360");

cmov2(135.0,pos2-2.0);
charstr("010");

cmov2(180.0,pos2-2.0);
charstr("020");

cmov2(225.0,pos2-2.0);
charstr("030");

cmov2(270.0,pos2-2.0);
charstr("040");

cmov2(315.0,pos2-2.0);
charstr("050");

cmov2(360.0,pos2-2.0);
charstr("060");

cmov2(405.0,pos2-2.0);
charstr("070");
```

```
cmov2(450.0,pos2-2.0);
charstr("080");

cmov2(495.0,pos2-2.0);
charstr("090");

cmov2(540.0,pos2-2.0);
charstr("100");

cmov2(585.0,pos2-2.0);
charstr("110");

cmov2(630.0,pos2-2.0);
charstr("120");

cmov2(675.0,pos2-2.0);
charstr("130");

cmov2(720.0,pos2-2.0);
charstr("140");

cmov2(765.0,pos2-2.0);
charstr("150");

cmov2(810.0,pos2-2.0);
charstr("160");

cmov2(855.0,pos2-2.0);
charstr("170");

cmov2(900.0,pos2-2.0);
charstr("180");

cmov2(945.0,pos2-2.0);
charstr("190");

cmov2(990.0,pos2-2.0);
charstr("200");

cmov2(1035.0,pos2-2.0);
charstr("210");

cmov2(1080.0,pos2-2.0);
charstr("220");

cmov2(1125.0,pos2-2.0);
charstr("230");

cmov2(1170.0,pos2-2.0);
charstr("240");

cmov2(1215.0,pos2-2.0);
```

```
charstr("250");

cmov2(1260.0,pos2-2.0);
charstr("260");

cmov2(1305.0,pos2-2.0);
charstr("270");

cmov2(1350.0,pos2-2.0);
charstr("280");

cmov2(1395.0,pos2-2.0);
charstr("290");

cmov2(1440.0,pos2-2.0);
charstr("300");

cmov2(1485.0,pos2-2.0);
charstr("310");

cmov2(1530.0,pos2-2.0);
charstr("320");

cmov2(1575.0,pos2-2.0);
charstr("330");

cmov2(1620.0,pos2-2.0);
charstr("340");

cmov2(1665.0,pos2-2.0);
charstr("350");

cmov2(1710.0,pos2-2.0);
charstr("360");

cmov2(1755.0,pos2-2.0);
charstr("010");

cmov2(1800.0,pos2-2.0);
charstr("020");

color(BLACK);

closeobj();

/*  Put all the pieces together  */

*heading_meter=genobj();
makeobj(*heading_meter);

/*  Draw the boundary */
```

```
callobj(meter);

/*  Draw the heading */

scrmask((int) pos1,(int) tempx,(int) pos2,(int) tempy);

pushmatrix();

transl1=gentag();
maketag(transl1);

translate(0.0,0.0,0.0);

callobj(theading);
scrmask(0,1023,0,767);
popmatrix();

color(RED);
linewidth(4);
move2(pos1+175.0/2,pos2);
draw2(pos1+175.0/2,tempy);

linewidth(1);


scrmask(0,1023,0,767);
closeobj();

}  /* makeheading */
```

E. OTHER.C

```
/*

This module contain the supporting routines for building the
scenery objects like the clouds and mountains.

*/

#include "road.h"

extern Dimension Roadlen, Roadwidth, Bendradius1;
extern Tag skylooktag, terrain1looktag;
extern Tag houselooktag, housetranstag;
extern Tag housescaletag;
extern Tag house1looktag, house1transtag;
extern Tag house1scaletag;
extern Angle Fov;

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

                    S K Y

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

makethesky(sky)
Object *sky;
{
*sky= genobj();
makeobj(*sky);

pushmatrix();
pushviewport();
viewport(0, 1023, 385, 767);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);

skylooktag = gentag();
maketag(skylooktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0);

pushmatrix();
translate(0.0, 100.0, 50000.0);
surf(0.0, 0.0, 0.0, 100000.0, 100000.0, SKY);
popmatrix();

popviewport();
popmatrix();
closeobj();
}  /* makethesky */
```

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

   CLOUDS AND MOUNTAINS

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

maketerrain1(terrain1)
Object *terrain1;
{
Dimension temp = -(Roadlen+500.0);
Dimension temp1 = -(Roadlen+500.0);
Dimension tempy = 0.0;
Dimension tempy1 = 100.0;
Dimension tempy2 = 350.0;

*terrain1= genobj();
makeobj(*terrain1);

/*  Generate some clouds  */

pushmatrix();
pushviewport();
viewport(0, 1023, 385, 767);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);
terrain1looktag= gentag():
maketag(terrain1looktag);
lookat(0.0. 0.0, 0.0, 0.0, 0.0, 0.0, 0);

pushmatrix();
color(WHITE);
translate(-1000.0, tempy1, temp);
scale(1.0, 1.0, 1.0);
rotate(-900, 'Y');
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(-1000.0, tempy1, temp);
scale(1.0, 0.8, 1.0);
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(-2000.0, tempy1, temp);
scale(2.0, 2.0, 1.0);
```

```
rotate(-900, 'Y');
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(-2000.0, tempy1, temp);
scale(2.0, 0.8, 1.0);
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0. 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(2000.0, tempy1, temp);
scale(3.0, 2.0, 1.0);
rotate(-900, 'Y');
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

pushmatrix();
color(WHITE);
translate(2000.0, tempy1, temp);
scale(2.0, 0.8, 1.0);
circf(0.0, 0.0, 40.0);
circf(50.0, 0.0, 30.0);
circf(40.0, 50.0, 40.0);
popmatrix();

/* Generate some mountains */

pushmatrix();
translate(-2000.0, tempy, temp);
scale (1.0, 0.1, 0.0);
color(MOUNTAIN1);
arcf(0.0, 0.0, 400.0, 0, 1800);
popmatrix();

pushmatrix();
translate(-1500.0, tempy, temp);
scale (1.0, 0.2, 0.0);
color(MOUNTAIN);
arcf(0.0, 0.0, 250.0, 0, 1800);
popmatrix();

pushmatrix();
translate(-1000.0, tempy, temp);
```

```
scale (1.0, 0.1. 0.0);
color(MOUNTAIN1);
arcf(0.0, 0.0, 300.0, 0, 1800);
popmatrix();

pushmatrix();
translate(1000.0, tempy, temp);
scale (1.0, 0.2. 0.0);
color(MOUNTAIN);
arcf(0.0, 0.0, 250.0, 0, 1800);
popmatrix();

pushmatrix();
translate(1500.0, tempy, temp);
scale (1.0, 0.1, 0.0);
color(MOUNTAIN1);
arcf(0.0, 0.0, 300.0, 0, 1800);
popmatrix();

pushmatrix();
translate(2000.0, tempy, temp);
scale (1.0, 0.1, 0.0);
color(MOUNTAIN1);
arcf(0.0, 0.0, 300.0, 0, 1800);
popmatrix();

popviewport();
popmatrix();
closeobj():
} /* maketerrain1*/

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

        BUILD  SURFACES

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

surf(x, y, z, width, length, roadcolor)
Coord x, y, z;
Dimension width, length;
Colorindex roadcolor;
{
Coord vertice[5][3];
Dimension temp;
temp = width/2;

vertice[0][0] = x;
vertice[0][1] = y;
vertice[0][2] = z;

vertice[1][0] = x - temp;
vertice[1][1] = y;
```

```
vertice[1][2] = z;

vertice[2][0] = x - temp;
vertice[2][1] = y;
vertice[2][2] = -length;

vertice[3][0] = x + temp;
vertice[3][1] = y;
vertice[3][2] = -length;

vertice[4][0] = x + temp;
vertice[4][1] = y;
vertice[4][2] = z;

color(roadcolor);
polf(5,vertice);
}   /* surf */

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

        B U I L D   R O A D   B E N D S

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

bend()
{
color(BLACK);
arcfi(0, 0, (int) Bendradius1, 900, 1800);
color(FIELD);
arcfi(0, 0, (int) (Bendradius1 - Roadwidth), 900, 1800);
}

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

        B U I L D   S I G N B O A R D

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

signb(width, length, height, bcolor)
Dimension width, length, height;
Colorindex bcolor;
{
Coord vertice[5][3];
Coord vertice1[5][3];
Dimension legwidth, temp1, temp2, temp3;
legwidth = 0.2;        /* size of supporting leg */
temp1 = length/2;
temp2 = length/4;
temp3 = legwidth/2;

vertice[0][0] = 0.0;
vertice[0][1] = height;
```

```
vertice[0][2] = 0.0;

vertice[1][0] = -temp1;
vertice[1][1] = height;
vertice[1][2] = 0.0;

vertice[2][0] = -temp1;
vertice[2][1] = width + height;
vertice[2][2] = 0.0;

vertice[3][0] = temp1;
vertice[3][1] = width + height;
vertice[3][2] = 0.0;

vertice[4][0] = temp1;
vertice[4][1] = height;
vertice[4][2] = 0.0;

color(bcolor);
polf(5,vertice);

/* Generate the supporting leg */
vertice1[0][0] = 0.0;
vertice1[0][1] = 0.0;
vertice1[0][2] = 0.0;

vertice1[1][0] = -temp3;
vertice1[1][1] = 0.0;
vertice1[1][2] = 0.0;

vertice1[2][0] = -temp3;
vertice1[2][1] = height;
vertice1[2][2] = 0.0;

vertice1[3][0] = temp3;
vertice1[3][1] = height;
vertice1[3][2] = 0.0;

vertice1[4][0] = temp3;
vertice1[4][1] = 0.0;
vertice1[4][2] = 0.0;

color(BLACK);
polf(5,vertice1);
}   /* signboard */

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

        B U I L D     A R R O W

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

```
polyarrow(bodywidth, headwidth, high, arrowcolor)
Colorindex arrowcolor;
Dimension bodywidth, headwidth, high;
{
Coord vertice[5][3], vertice1[3][3];
Dimension bodyheight = 0.8;
Dimension headheight = 1.5;
Dimension temp1 = bodywidth/2;
Dimension temp2 = headwidth/2;

vertice[0][0] = 0.0;
vertice[0][1] = 0.0 + high;
vertice[0][2] = 0.0;

vertice[1][0] = -temp1;
vertice[1][1] = 0.0 + high;
vertice[1][2] = 0.0;

vertice[2][0] = -temp1;
vertice[2][1] = bodyheight + high;
vertice[2][2] = 0.0;

vertice[3][0] = temp1;
vertice[3][1] = bodyheight + high;
vertice[3][2] = 0.0;

vertice[4][0] = temp1;
vertice[4][1] = 0.0 + high;
vertice[4][2] = 0.0;

color(arrowcolor);
polf(5,vertice);

vertice1[0][0] = -temp2;
vertice1[0][1] = bodyheight + high;
vertice1[0][2] = 0.0;

vertice1[1][0] = 0.0;
vertice1[1][1] = headheight + high;
vertice1[1][2] = 0.0;

vertice1[2][0] = temp2;
vertice1[2][1] = bodyheight + high;
vertice1[2][2] = 0.0;

color(arrowcolor);
polf(3,vertice1);
} /* polyarrow */

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

        BUILD    HOUSE
```

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

makehouse(house)
Object *house;
{
float sidewall[5][2], roof[4][2], chmwall1[4][2];
float chmwall2[4][2], sideroof[4][2];

*house=genobj();
makeobj(*house);

pushmatrix();
pushviewport();
viewport(0, 1023, 385, 767);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);
houselooktag = gentag();
maketag(houselooktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0. 0);

pushmatrix():
housetranstag = gentag();
maketag(housetranstag):
translate(0.0, 0.0, 0.0);
housescaletag = gentag();
maketag(housescaletag);
scale(1.0, 1.0, 1.0);

/* Draw front wall */

color(WALL);
rectf(-1.0,0.0,16.0,10.0);

/* Draw side wall */

sidewall[0][0]=(-4.0);
sidewall[0][1]=(2.0);
sidewall[1][0]=(0.0);
sidewall[1][1]=(0.0);
sidewall[2][0]=(0.0);
sidewall[2][1]=(10.0);
sidewall[3][0]=(-3.0);
sidewall[3][1]=(13.0);
sidewall[4][0]=(-4.0);
sidewall[4][1]=(11.5);

color(SIDEWALL);
polf2(5,sidewall);

/* Draw roof and sideroof */

roof[0][0]=(-1.0);
```

```
roof[0][1]=(10.0);
roof[1][0]=(17.0);
roof[1][1]=(10.0);
roof[2][0]=(14.0);
roof[2][1]=(13.5);
roof[3][0]=(-3.0);
roof[3][1]=(13.5);

color(ROOF);
polf2(4,roof);

sideroof[0][0]=(-4.3);
sideroof[0][1]=(11.5);
sideroof[1][0]=(-4.0);
sideroof[1][1]=(11.5);
sideroof[2][0]=(-2.8);
sideroof[2][1]=(13.1);
sideroof[3][0]=(-3.0);
sideroof[3][1]=(13.5);

color(SIDEROOF);
polf2(4,sideroof);

/* Draw window */

color(WINDOW);
rectf(2.0,4.0,5.0,7.0);
rectf(9.0,4.0,12.0,7.0);

/* Draw window frames */

color(FRAME);
linewidth(4);
move(2.0,4.0,0.0);
draw(5.0,4.0,0.0);
draw(5.0,7.0,0.0);
draw(2.0,7.0,0.0);
draw(2.0,4.0,0.0);
move(3.5,4.0,0.0);
draw(3.5,7.0,0.0);
move(2.0,5.5,0.0);
draw(5.0,5.5,0.0);

move(9.0,4.0,0.0);
draw(12.0,4.0,0.0);
draw(12.0,7.0,0.0);
draw(9.0,7.0,0.0);
draw(9.0,4.0,0.0);
move(10.5,4.0,0.0);
draw(10.5,7.0,0.0);
move(9.0,5.5,0.0);
draw(12.0,5.5,0.0);
```

```
/* Draw chimney front wall */

color(SIDEWALL);
rectf(1.0,12.0,3.0,14.2);

/* Draw the hole on the chimney */

color(BLACK);
rectf(1.5,13.3,2.5,13.8);

/* Draw top and side walls of the chimney */

chmwall1[0][0]=0.5;
chmwall1[0][1]=12.5;
chmwall1[1][0]=1.0;
chmwall1[1][1]=12.0;
chmwall1[2][0]=1.0;
chmwall1[2][1]=14.2;
chmwall1[3][0]=0.5;
chmwall1[3][1]=14.7;

color(CHMWALL1);
polf2(4,chmwall1);

chmwall2[0][0]=2.5;
chmwall2[0][1]=14.7;
chmwall2[1][0]=3.0;
chmwall2[1][1]=14.2;
chmwall2[2][0]=1.0;
chmwall2[2][1]=14.2;
chmwall2[3][0]=0.5;
chmwall2[3][1]=14.7;

color(CHMWALL2);
polf2(4,chmwall2);
popmatrix();

popviewport();
popmatrix();
closeobj();
}  /* makehouse */

makehouse1(house1)
Object *house1;
{
float sidewall[5][2], roof[4][2], chmwall1[4][2];
float chmwall2[4][2], sideroof[4][2];

*house1=genobj();
makeobj(*house1);

pushmatrix():
```

```
pushviewport();
viewport(0, 1023, 385, 767);
setdepth(0,1023);
perspective(Fov, 1023.0/385.0, 0.0, 1023.0);
house1looktag = gentag();
maketag(house1looktag);
lookat(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0);

pushmatrix();
house1transtag = gentag();
maketag(house1transtag);
translate(0.0, 0.0, 0.0);
house1scaletag = gentag();
maketag(house1scaletag);
scale(1.0, 1.0, 1.0);

/* Draw front wall */

color(WALL1);
rectf(-1.0,0.0,16.0,10.0);

/* Draw side wall */

sidewall[0][0]=(-4.0);
sidewall[0][1]=(2.0);
sidewall[1][0]=(0.0);
sidewall[1][1]=(0.0);
sidewall[2][0]=(0.0);
sidewall[2][1]=(10.0);
sidewall[3][0]=(-3.0);
sidewall[3][1]=(13.0);
sidewall[4][0]=(-4.0);
sidewall[4][1]=(11.5);

color(SIDEWALL1);
polf2(5,sidewall);

/* Draw roof and sideroof */

roof[0][0]=(-1.0);
roof[0][1]=(10.0);
roof[1][0]=(17.0);
roof[1][1]=(10.0);
roof[2][0]=(14.0);
roof[2][1]=(13.5);
roof[3][0]=(-3.0);
roof[3][1]=(13.5);

color(ROOF1);
polf2(4,roof);

sideroof[0][0]=(-4.3);
```

```
sideroof[0][1]=(11.5);
sideroof[1][0]=(-4.0);
sideroof[1][1]=(11.5);
sideroof[2][0]=(-2.8);
sideroof[2][1]=(13.1);
sideroof[3][0]=(-3.0);
sideroof[3][1]=(13.5);

color(SIDEROOF);
polf2(4,sideroof);

/* Draw window */

color(WINDOW);
rectf(2.0,4.0,5.0,7.0);
rectf(9.0,4.0,12.0,7.0);

/* Draw window frames */

color(FRAME);
linewidth(4);
move(2.0,4.0,0.0);
draw(5.0,4.0,0.0);
draw(5.0,7.0,0.0);
draw(2.0,7.0,0.0);
draw(2.0,4.0,0.0);
move(3.5,4.0,0.0);
draw(3.5,7.0,0.0);
move(2.0,5.5,0.0);
draw(5.0,5.5,0.0);

move(9.0,4.0,0.0);
draw(12.0,4.0,0.0);
draw(12.0,7.0,0.0);
draw(9.0,7.0,0.0);
draw(9.0,4.0,0.0);
move(10.5,4.0,0.0);
draw(10.5,7.0,0.0);
move(9.0,5.5,0.0);
draw(12.0,5.5,0.0);

/* Draw chimney front wall */

color(SIDEWALL1);
rectf(1.0,12.0,3.0,14.2);

/* Draw the hole on the chimney */

color(BLACK);
rectf(1.5,13.3,2.5,13.8);

/* Draw top and side walls of the chimney */
```

```
chmwall1[0][0]=0.5;
chmwall1[0][1]=12.5;
chmwall1[1][0]=1.0;
chmwall1[1][1]=12.0;
chmwall1[2][0]=1.0;
chmwall1[2][1]=14.2;
chmwall1[3][0]=0.5;
chmwall1[3][1]=14.7;

color(CHMWALL1);
polf2(4,chmwall1);

chmwall2[0][0]=2.5;
chmwall2[0][1]=14.7;
chmwall2[1][0]=3.0;
chmwall2[1][1]=14.2;
chmwall2[2][0]=1.0;
chmwall2[2][1]=14.2;
chmwall2[3][0]=0.5;
chmwall2[3][1]=14.7;

color(CHMWALL2);
polf2(4,chmwall2);
popmatrix();

popviewport();
popmatrix();
closeobj();
}  /* makehouse1 */
```

F. HELP.C

```
/*

This module creates the welcome and help screen.

*/

#include "road.h"

static int parray[4][2] = {{275,600},{250,625},{275,625},{300,600}};
static int parray1[4][2] = {{275,475},{250,500},{275,500},{300,475}};

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

        W E L C O M E     D I S P L A Y

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

welcome()
{

/* Loop until we get a mouse button hit */

    color(YELLOW);
    clear();

    color(BLUE);

    rectfi(200,625,300,700);
    rectfi(200,600,225,625);
    polf2i(4,parray);

    rectfi(325,600,425,700);
    rectfi(450,600,550,700);
    rectfi(575,600,675,700);
    polf2i(4,parray1);

    rectfi(200,475,225,500);
    rectfi(200,500,300,575);
    rectfi(325,475,425,575);
    rectfi(450,475,550,500);
    rectfi(450,500,475,575);
    rectfi(575,475,675,500);
    rectfi(575,500,600,575);
    rectfi(700,525,800,575);
    rectfi(737,475,762,525);

    color(YELLOW);

    rectfi(225,650,275,675);
    rectfi(350,625,400,675);
    rectfi(475,600,525,625);
    rectfi(475,650,525,675);
    rectfi(575,625,600,675);
    rectfi(625,625,650,675);
    rectfi(225,525,275,550);
    rectfi(350,525,400,550);
    rectfi(350,475.400,500);
    rectfi(725,550,775,575);

    color(BLACK);

    cmov2i(200,350);
    charstr("Welcome to the world of ROAD RALLY");

    cmov2i(200,325);
    charstr("You drive a car on a road controlling");
    charstr(" its speed and direction with the");

    cmov2i(200,300);
    charstr("mouse. To exit the program"):
    charstr(" at any time press all three mouse simultaneously.");

    cmov2i(200,275);
    charstr("After the bell ring, to continue with the");
    charstr(" program press the left mouse button.");

    cmov2i(200,250);
    charstr("HELP is available by pressing the keyboard key h");
    charstr(" while the car is moving.");

    linewidth(5);
    cmov2i(200,175);
    charstr("Author : Tan Chiam Huat");
    color(RED);
    cmov2i(200,150);
    charstr("This image is contributed by: Mike Whiting");
    color(BLACK);
    linewidth(1);
    swapbuffers();

} /* welcome */

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

        HELP    DISPLAY

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

help()
{
Icoord x  = 100;
Icoord y  = 340;
Icoord iy = 22;

/* Loop until we get a mouse button hit */

while (getbutton(MOUSE1) == 0 && getbutton(MOUSE2) == 0 &&
        getbutton(MOUSE3) == 0)
    {

    pushmatrix();
    pushviewport();
    viewport(0, 1023, 0, 380);
    ortho2(0.0, 1023.0, 0.0, 380.0);
    color(WHITE);
    clear();

    color(BLACK);

    linewidth(5);
    cmov2i(x, y);
    charstr("HELP INFORMATION: Press any mouse button to continue");

    cmov2i(x, y - iy);
    charstr("KEY                        REMARK");

    cmov2i(x, y - 2 * iy);
    charstr("a or A or Left button    : Accelerate");

    cmov2i(x, y - 3 * iy);
    charstr("b or B or Middle button  : Brake");

    cmov2i(x, y - 4 * iy);
    charstr("c or C                   : Clock switch");

    cmov2i(x, y - 5 * iy);
    charstr("e or E or Right button   : Emergence Stop");

    cmov2i(x, y - 6 * iy);
    charstr("h or H                   : Help");

    cmov2i(x, y - 7 * iy);
    charstr("o or O                   : Odometer reset");

    cmov2i(x, y - 8 * iy);
    charstr("q or Q                   : Autopilot");

    cmov2i(x, y - 9 * iy);
    charstr("s or S                   : Sound danger");

    cmov2i(x, y - 10 * iy);
    charstr("t or T                   : Timer (Integration)");
```

```
     cmov2i(x, y - 11 * iy);
     charstr("z or Z                : Information");

     cmov2i(x, y - 13 * iy);
     charstr("To TURN LEFT           : Move mouse to the left");

     cmov2i(x, y - 14 * iy);
     charstr("To TURN RIGHT          : Move mouse to the right");

     linewidth(1);
     popviewport();
     popmatrix();
     swapbuffers();

     } /* while */

} /* help */
```

G. LETTER.C

```
/* This routine is written for the IRIS-2400
   This is routine letter.c...

   This file supports routine title.c, which constructs the
   title page of the font building utility "BUILDFONT."

   This file contains routines to display block alphabetic characters
   suitable for inclusion into graphics objects.  These letters are
   used instead of IRIS FONTS when one desires to treat them as
   graphics objects that can be rotated, scaled, etc. (font char-
   acters can't)

   This file includes routines for 27 characters, "A" through "Z",
   and also ":" and " " (blank) (but not "G","Q","V","W","X",Z")

   The routine draws the desired letter in absolute coordinates.
   in the center of the display.

   To use these routines, the color desired for the letter must
   be specified when the object is created (in the user program),
   and the desired backgound color must be passed to the routine.

   Original version written by J. Artero and R. Kirsch; current
   version written by L. Williamson

*/

#include "gl.h"
#include "device.h"

letter(asci,backcolor)

int asci;                  /* index of character we want to display  */

Colorindex backcolor;       /* specified background color */

{

   Coord box [8][2];        /* vector of coordinates forming the
                              vertices of a letter object */


   switch(asci)
   {

      case 'A':

         box[0][0]=4.6875;
         box[0][1]=3.25;
```

```
    box[1][0]=4.9375;
    box[1][1]=4.25;
    box[2][0]=5.0625;
    box[2][1]=4.25;
    box[3][0]=5.3125;
    box[3][1]=3.25;
    polf2(4,box);

    color(backcolor);
    box[0][0]=4.8125;
    box[0][1]=3.25;
    box[1][0]=4.84375;
    box[1][1]=3.375;
    box[2][0]=5.15625;
    box[2][1]=3.375;
    box[3][0]=5.1875;
    box[3][1]=3.25;
    polf2(4,box);

    box[0][0]=4.875;
    box[0][1]=3.5;
    box[1][0]=5.0;
    box[1][1]=4.0;
    box[2][0]=5.125;
    box[2][1]=3.5;
    polf2(3,box);

    break;

  case ’B’:

    box[0][0]=4.6875;
    box[0][1]=3.25;
    box[1][0]=4.6875;
    box[1][1]=4.25;
    box[2][0]=5.1875;
    box[2][1]=4.25;
    box[3][0]=5.3125;
    box[3][1]=4.125;
    box[4][0]=5.3125;
    box[4][1]=3.375;
    box[5][0]=5.1875;
    box[5][1]=3.25;
    polf2(6,box);

    color(backcolor);
    box[0][0]=5.25;
    box[0][1]=3.8125;
    box[1][0]=5.3125;
    box[1][1]=3.875;
    box[2][0]=5.3125;
    box[2][1]=3.75;
  polf2(3,box);

  box[0][0]=4.8125;
  box[0][1]=3.375;
  box[1][0]=4.8125;
  box[1][1]=3.75;
  box[2][0]=5.125;
  box[2][1]=3.75;
  box[3][0]=5.1875;
  box[3][1]=3.6875;
  box[4][0]=5.1875;
  box[4][1]=3.4375;
  box[5][0]=5.125;
  box[5][1]=3.375;
  polf2(6,box);

  box[0][0]=4.8125;
  box[0][1]=3.875;
  box[1][0]=4.8125;
  box[1][1]=4.125;
  box[2][0]=5.125;
  box[2][1]=4.125;
  box[3][0]=5.1875;
  box[3][1]=4.0625;
  box[4][0]=5.1875;
  box[4][1]=3.9375;
  box[5][0]=5.125;
  box[5][1]=3.875;
  polf2(6,box);

  break;

case 'C':

   box[0][0]=4.6875;
   box[0][1]=3.375;
   box[1][0]=4.6875;
   box[1][1]=4.125;
   box[2][0]=4.8125;
   box[2][1]=4.25;
   box[3][0]=5.1875;
   box[3][1]=4.25;
   box[4][0]=5.3125;
   box[4][1]=4.125;
   box[5][0]=5.3125;
   box[5][1]=3.375;
   box[6][0]=5.1875;
   box[6][1]=3.25;
   box[7][0]=4.8125;
   box[7][1]=3.25;
   polf2(8,box);
```

```
      color(backcolor);
      box[0][0]=4.8125;
      box[0][1]=3.4375;
      box[1][0]=4.8125;
      box[1][1]=4.0625;
      box[2][0]=4.875;
      box[2][1]=4.125;
      box[3][0]=5.125;
      box[3][1]=4.125;
      box[4][0]=5.1875;
      box[4][1]=4.0625;
      box[5][0]=5.1875;
      box[5][1]=3.4375;
      box[6][0]=5.125;
      box[6][1]=3.375;
      box[7][0]=4.875;
      box[7][1]=3.375;
      polf2(8,box);

      rectf(5.1875,3.5,5.3125,4.00);

    break;

  case ’D’:

      box[0][0]=4.6875;
      box[0][1]=3.25;
      box[1][0]=4.6875;
      box[1][1]=4.25;
      box[2][0]=5.1875;
      box[2][1]=4.25;
      box[3][0]=5.3125;
      box[3][1]=4.125;
      box[4][0]=5.3125;
      box[4][1]=3.375;
      box[5][0]=5.1875;
      box[5][1]=3.25;
      polf2(6,box);

      color(backcolor);
      box[0][0]=4.8125;
      box[0][1]=3.375;
      box[1][0]=4.8125;
      box[1][1]=4.125;
      box[2][0]=5.125;
      box[2][1]=4.125;
      box[3][0]=5.1875;
      box[3][1]=4.0625;
      box[4][0]=5.1875;
      box[4][1]=3.4375;
      box[5][0]=5.125;
      box[5][1]=3.375;
    polf2(6,box);

    break;

  case 'E':

    rectf(4.6875,4.125,5.25,4.25);
    rectf(4.6875,3.25,5.3125,3.375);
    rectf(4.6875,3.25,4.8125,4.25);
    rectf(4.8125,3.75,5.0625,3.875);

    break;

  case 'F':

    rectf(4.6875,3.25,4.8125,4.25);
    rectf(4.6875,4.125,5.3125,4.25);
    rectf(4.8125,3.75,5.125,3.875);

    break;

  case 'H':

    rectf(4.6875,3.25,4.8125,4.25);
    rectf(4.8125,3.6875,5.1875,3.8125);
    rectf(5.1875,3.25,5.3125,4.25);

    break;

  case 'I':

    rectf(4.6875,4.125,5.3125,4.25);
    rectf(4.6875,3.25,5.3125,3.375);
    rectf(4.9375,3.25,5.0625,4.25);

    break;

  case 'J':

    box[0][0]=4.6875;
    box[0][1]=3.375;
    box[1][0]=4.6875;
    box[1][1]=3.625;
    box[2][0]=5.3125;
    box[2][1]=3.625;
    box[3][0]=5.3125;
    box[3][1]=3.375;
    box[4][0]=5.1875;
    box[4][1]=3.25;
    box[5][0]=4.8125;
    box[5][1]=3.25;
    polf2(6,box);
```

```
   rectf(5.2,3.625,5.3125,4.25);

   color(backcolor);
   box[0][0]=4.8125;
   box[0][1]=3.4375;
   box[1][0]=4.8125;
   box[1][1]=3.625;
   box[2][0]=5.1875;
   box[2][1]=3.625;
   box[3][0]=5.1875;
   box[3][1]=3.4375;
   box[4][0]=5.125;
   box[4][1]=3.375;
   box[5][0]=4.875;
   box[5][1]=3.375;
   polf2(6,box);

   break;

case 'K':

  rectf(4.6875,3.25,5.3125,4.25);

   color(backcolor);
   box[0][0]=4.8125;
   box[0][1]=3.875;
   box[1][0]=4.8125;
   box[1][1]=4.25;
   box[2][0]=5.125;
   box[2][1]=4.25;
   polf2(3,box);

   box[0][0]=5.02;
   box[0][1]=3.875;
   box[1][0]=5.3125;
   box[1][1]=4.25;
   box[2][0]=5.3125;
   box[2][1]=3.25;
   polf2(3,box);

   box[0][0]=4.8125;
   box[0][1]=3.25;
   box[1][0]=4.8125;
   box[1][1]=3.625;
   box[2][0]=4.9;
   box[2][1]=3.74;
   box[3][0]=5.14;
   box[3][1]=3.25;
   polf2(4,box);

   break;
```

```
case 'L':

    rectf(4.6875,3.25,4.8125,4.25);
    rectf(4.6875,3.25,5.3125,3.375);

    break;

case 'M':

    rectf(4.6875,3.25,5.3125,4.25);

    color(backcolor);
    box[0][0]=4.6875;
    box[0][1]=4.25;
    box[1][0]=5.3125;
    box[1][1]=4.25;
    box[2][0]=5.0;
    box[2][1]=3.75;
    polf2(3,box);

    box[0][0]=4.8125;
    box[0][1]=3.25;
    box[1][0]=4.8125;
    box[1][1]=3.8125;
    box[2][0]=5.125;
    box[2][1]=3.25;
    polf2(3,box);

    box[0][0]=4.875;
    box[0][1]=3.25;
    box[1][0]=5.1875;
    box[1][1]=3.8125;
    box[2][0]=5.1875;
    box[2][1]=3.25;
    polf2(3,box);

    break;

case 'N':

    rectf(4.6875,3.25,5.3125,4.25);

    color(backcolor);
    box[0][0]=4.8125;
    box[0][1]=3.25;
    box[1][0]=4.8125;
    box[1][1]=3.9375;
    box[2][0]=5.1875;
    box[2][1]=3.25;
    polf2(3,box);

    box[0][0]=4.8125;
    box[0][1]=4.25;
    box[1][0]=5.1875;
    box[1][1]=4.25;
    box[2][0]=5.1875;
    box[2][1]=3.5625;
    polf2(3,box);

    break;

case 'O':

    box[0][0]=4.6875;
    box[0][1]=3.375;
    box[1][0]=4.6875;
    box[1][1]=4.125;
    box[2][0]=4.8125;
    box[2][1]=4.25;
    box[3][0]=5.1875;
    box[3][1]=4.25;
    box[4][0]=5.3125;
    box[4][1]=4.125;
    box[5][0]=5.3125;
    box[5][1]=3.375;
    box[6][0]=5.1875;
    box[6][1]=3.25;
    box[7][0]=4.8125;
    box[7][1]=3.25;
    polf2(8,box);

    color(backcolor);
    box[0][0]=4.8125;
    box[0][1]=3.4375;
    box[1][0]=4.8125;
    box[1][1]=4.0625;
    box[2][0]=4.875;
    box[2][1]=4.125;
    box[3][0]=5.125;
    box[3][1]=4.125;
    box[4][0]=5.1875;
    box[4][1]=4.0625;
    box[5][0]=5.1875;
    box[5][1]=3.4375;
    box[6][0]=5.125;
    box[6][1]=3.375;
    box[7][0]=4.875;
    box[7][1]=3.375;
    polf2(8,box);

    break;

case 'P':
    box[0][0]=4.6875;
    box[0][1]=3.25;
    box[1][0]=4.6875;
    box[1][1]=4.25;
    box[2][0]=5.1875;
    box[2][1]=4.25;
    box[3][0]=5.3125;
    box[3][1]=4.125;
    box[4][0]=5.3125;
    box[4][1]=3.25;
    polf2(5,box);

    color(backcolor);
    box[0][0]=4.8125;
    box[0][1]=3.25;
    box[1][0]=5.3125;
    box[1][1]=3.8125;
    box[2][0]=5.3125;
    box[2][1]=3.25;
    polf2(3,box);

    box[0][0]=4.8125;
    box[0][1]=3.8125;
    box[1][0]=4.8125;
    box[1][1]=4.125;
    box[2][0]=5.125;
    box[2][1]=4.125;
    box[3][0]=5.1875;
    box[3][1]=4.0625;
    box[4][0]=5.1875;
    box[4][1]=3.875;
    box[5][0]=5.125;
    box[5][1]=3.8125;
    polf2(6,box);

    rectf(4.8125,3.25,5.3125,3.6875);

    break;

case ’R’:

    box[0][0]=4.6875;
    box[0][1]=3.25;
    box[1][0]=4.6875;
    box[1][1]=4.25;
    box[2][0]=5.1875;
    box[2][1]=4.25;
    box[3][0]=5.3125;
    box[3][1]=4.125;
    box[4][0]=5.3125;
    box[4][1]=3.25;
    polf2(5,box);
    color(backcolor);
    box[0][0]=5.1875;
    box[0][1]=3.625;
    box[1][0]=5.3125;
    box[1][1]=3.75;
    box[2][0]=5.3125;
    box[2][1]=3.25;
    polf2(3,box);

    box[0][0]=4.8125;
    box[0][1]=3.75;
    box[1][0]=4.8125;
    box[1][1]=4.125;
    box[2][0]=5.125;
    box[2][1]=4.125;
    box[3][0]=5.1875;
    box[3][1]=4.0625;
    box[4][0]=5.1875;
    box[4][1]=3.8125;
    box[5][0]=5.125;
    box[5][1]=3.75;
    polf2(6,box);

    box[0][0]=4.8125;
    box[0][1]=3.25;
    box[1][0]=4.8125;
    box[1][1]=3.625;
    box[2][0]=5.05;
    box[2][1]=3.625;
    box[3][0]=5.175;
    box[3][1]=3.25;
    polf2(4,box);

    break;

case 'S':

    box[0][0]=4.6875;
    box[0][1]=3.375;
    box[1][0]=4.6875;
    box[1][1]=4.125;
    box[2][0]=4.8125;
    box[2][1]=4.25;
    box[3][0]=5.1875;
    box[3][1]=4.25;
    box[4][0]=5.3125;
    box[4][1]=4.125;
    box[5][0]=5.3125;
    box[5][1]=3.375;
    box[6][0]=5.1875;
    box[6][1]=3.25;
    box[7][0]=4.8125;
```

```
box[7][1]=3.25;
polf2(8,box);

color(backcolor);
box[0][0]=4.8125;
box[0][1]=3.4375;
box[1][0]=4.8125;
box[1][1]=3.75;
box[2][0]=5.125;
box[2][1]=3.75;
box[3][0]=5.1875;
box[3][1]=3.6875;
box[4][0]=5.1875;
box[4][1]=3.4375;
box[5][0]=5.125;
box[5][1]=3.375;
box[6][0]=4.875;
box[6][1]=3.375;
polf2(7,box);

box[0][0]=4.8125;
box[0][1]=3.9375;
box[1][0]=4.8125;
box[1][1]=4.0625;
box[2][0]=4.875;
box[2][1]=4.125;
box[3][0]=5.125;
box[3][1]=4.125;
box[4][0]=5.1875;
box[4][1]=4.0625;
box[5][0]=5.1875;
box[5][1]=3.875;
box[6][0]=4.875;
box[6][1]=3.875;
polf2(7,box);

box[0][0]=4.6875;
box[0][1]=3.5625;
box[1][0]=4.6875;
box[1][1]=3.875;
box[2][0]=4.8125;
box[2][1]=3.75;
box[3][0]=4.8125;
box[3][1]=3.5625;
polf2(4,box);

box[0][0]=5.1875;
box[0][1]=3.875;
box[1][0]=5.1875;
box[1][1]=4.0;
box[2][0]=5.3125;
box[2][1]=4.0;
```

```
   box[3][0]=5.3125;
   box[3][1]=3.75;
   polf2(4,box);

   break;

case 'T':

   rectf(4.6875,4.125,5.3125,4.25);
   rectf(4.9375,3.25,5.0625,4.25);
   break;

case 'U':

   box[0][0]=4.6875;
   box[0][1]=3.375;
   box[1][0]=4.6875;
   box[1][1]=4.25;
   box[2][0]=5.3125;
   box[2][1]=4.25;
   box[3][0]=5.3125;
   box[3][1]=3.25;
   box[4][0]=4.8125;
   box[4][1]=3.25;
   polf2(5,box);

   color(backcolor);
   box[0][0]=4.8125;
   box[0][1]=3.4375;
   box[1][0]=4.8125;
   box[1][1]=4.25;
   box[2][0]=5.1875;
   box[2][1]=4.25;
   box[3][0]=5.1875;
   box[3][1]=3.5325;
   box[4][0]=5.01;
   box[4][1]=3.375;
   box[5][0]=4.875;
   box[5][1]=3.375;
   polf2(6,box);

   box[0][0]=5.0625;
   box[0][1]=3.25;
   box[1][0]=5.1875;
   box[1][1]=3.375;
   box[2][0]=5.1875;
   box[2][1]=3.25;
   polf2(3,box);

   break;

case 'Y':
      box[0][0]=4.6875;
      box[0][1]=4.25;
      box[1][0]=4.9375;
      box[1][1]=3.75;
      box[2][0]=5.0625;
      box[2][1]=3.75;
      box[3][0]=4.8125;
      box[3][1]=4.25;
      polf2(4,box);


      box[0][0]=4.9375;
      box[0][1]=3.75;
      box[1][0]=5.0625;
      box[1][1]=3.75;
      box[2][0]=5.3125;
      box[2][1]=4.25;
      box[3][0]=5.1875;
      box[3][1]=4.25;
      polf2(4,box);

      rectf(4.9375,3.25,5.0625,3.75);

      break;

    case ':':

      rectf(4.9375,3.35,5.0625,3.60);
      rectf(4.9375,3.90,5.0625,4.15);

      break;

    case ' ':

      break;

  } /* end  switch */

}    /* end routine "letter" */
```

H. FIND_SUBGOAL.C

```
/*

Look for the next subgoal along the road

*/

#include "road.h"

static Boolean start = FALSE;

find_subgoal(roadmap, no_coord, where, tolerance, pred_distance, vx, vy, vz)
float pred_distance;
float roadmap[][3];
float tolerance;
float vx, vy, vz;
int no_coord, where;
{
        float dist, temp;
        float x, y, z;
        int i;

        for (i = where; i < no_coord; ++i)
                {
                x = roadmap[i][0] - vx;
                y = roadmap[i][1] - vy;
                z = roadmap[i][2] - vz;
                dist = sqrt(x*x + y*y);
                temp = pred_distance - dist;

                /* converts negative to positive */
                if (temp < 0)
                        temp = -(temp);

                if (!start)
                        {

                        /* This works only when autopilot is turned
                        on for the first time on the first stretch
                        of the cicuit.  Problem if otherwise.  */

                        if (temp <= tolerance && roadmap[i][1] > vy)
                                {
                                start = TRUE;
                                return(i);
                                }
                        }
                else
                        if (temp <= tolerance)
                                {
```

```
                    start = TRUE;
                    return(i);
                    }
            }

        /* If no points found, return an error code */
        return(-1);

}  /* find_subgoal */
```

I. MAP.C

```
/*

This module works independently from the rest of the system.
This module generate the road map for autonomous navgiation.

*/

#include <stdio.h>
#include <math.h>

main()
{
    FILE *fp;
    int i;
    /*  Road Specification */
    /* Note: Must match that used in the carsimu.c program */
    float bendradius = 80.0;
    float roadwidth = 16.0;
    float len1 = 400.0;
    float len2 = 400.0;
    float len3 = 400.0;
    float len4 = 400.0;
    float newx, newy, miss;
    float calx, caly, start_rad;
    float perstep_rad;
    float step = 1.0;     /* road map increment step */
    float rad1 = bendradius - roadwidth/2;
    float rad2 = bendradius - roadwidth/2;
    float rad3 = bendradius - roadwidth/2;
    float lastxvalue;
    float lastyvalue;
    float x1, y1, z1;
    float x2, y2, z2;
    float x3, y3, z3;
    float x4, y4, z4;
    float x5, y5, z5;
    float x6, y6, z6;
    float x7, y7, z7;
    float x8, y8, z8;

    /* Road Segment Specifications */
    x1 = 0.0; y1 = 0.0; z1 = 0.0;
    x2 = 0.0; y2 = len1; z2 = 0.0;
    x3 = rad1; y3 = y2 + rad1; z3 = 0.0;
    x4 = x3 + len2; y4 = y3; z4 = 0.0;
    x5 = x4 + rad2; y5 = y4 - rad2; z5 = 0.0;
    x6 = x5; y6 = y5 - len3; z6 = 0.0;
    x7 = x5 - rad3; y7 = y6 - rad3; z7 = 0.0;
    x8 = x7 - len4; y8 = y7; z8 = 0.0;
```

```
        fp = fopen("roadmap","w");

        newy = y1;
        for (i = 0; newy <= y2; ++i)
                {
#ifdef DEBUG
                printf("%.2f %.2f %.2f\n",x1,newy,z1);
#endif
                fprintf(fp,"%.2f %.2f %.2f\n",x1,newy,z1);
                lastyvalue = newy;
                newy += step;
                }
        newy = lastyvalue;
        miss = y2 - newy;
#ifdef DEBUG
        printf("miss1 %.2f\n",miss);
#endif

        start_rad = 0;
        if (miss > 0)
                {
                start_rad = miss/rad1;
                calx = cos(start_rad);
                caly = sin(start_rad);
                newy += caly;
                newx += calx;
#ifdef DEBUG
                printf("%.2f %.2f %.2f\n",x2+newx,y2+newy,z2);
#endif
                fprintf(fp,"%.2f %.2f %.2f\n",x2+newx,y2+newy,z2);
                }

        perstep_rad = step/rad1;
        for (i = 0; newx <= x3; ++i)
                {
                start_rad += perstep_rad;
                calx = rad1 * cos(start_rad);
                caly = rad1 * sin(start_rad);
                lastxvalue = newx;
                lastyvalue = newy;
                newy = y2 + caly;
                newx = x2 + (rad1 - calx);
                if (newx < x3)
                        {
#ifdef DEBUG
                        printf("%.2f %.2f %.2f\n",newx,newy,z2);
#endif
                        fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z2);
                        }
                }

        newx = lastxvalue;
        newy = lastyvalue;
        miss = x3 - newx;
#ifdef DEBUG
        printf("miss2 %.2f\n",miss);
#endif

        if (miss > 0)
                {
                newx = x3 + miss;
#ifdef DEBUG
                printf("%.2f %.2f %.2f\n",newx,y4,z3);
#endif
                fprintf(fp,"%.2f %.2f %.2f\n",newx,y4,z3);
                }

        for (i = 0; newx <= x4; ++i)
                {
                lastxvalue = newx;
                newx += step;
                if (newx <= x4)
                        {
#ifdef DEBUG
                        printf("%.2f %.2f %.2f\n",newx,y4,z3);
#endif
                        fprintf(fp,"%.2f %.2f %.2f\n",newx,y4,z3);
                        }
                }
        newx = lastxvalue;
        miss = x4 - newx;
#ifdef DEBUG
        printf("miss3 %.2f\n",miss);
#endif

        start_rad = 0;
        if (miss > 0)
                {
                start_rad = miss/rad2;
                caly = rad2 * cos(start_rad);
                calx = rad2 * sin(start_rad);
                newy = y4 - (rad2 - caly);
                newx = x4 + calx;
#ifdef DEBUG
                printf("%.2f %.2f %.2f\n",newx,newy,z4);
#endif
                fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z4);
                }

        perstep_rad = step/rad1;
        for (i = 0; newy >= y5; ++i)
                {
                start_rad += perstep_rad;
                caly = rad2 * cos(start_rad);
        calx = rad2 * sin(start_rad);
        lastxvalue = newx;
        lastyvalue = newy;
        newx = x4 + calx;
        newy = y4 - (rad2 - caly);
        if (newy >= y5)
            {
#ifdef DEBUG
            printf("%.2f %.2f %.2f\n",newx,newy,z4);
#endif
            fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z4);
            }
        }

    newx = lastxvalue;
    newy = lastyvalue;
    miss = newy - y5;
#ifdef DEBUG
    printf("miss4 %.2f\n",miss);
#endif

    if (miss > 0)
        {
        newy = y5 + miss;
#ifdef DEBUG
        printf("%.2f %.2f %.2f\n",newx,newy,z5);
#endif
        fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z5);
        }

    for (i = 0; newy >= y6; ++i)
        {
        lastyvalue = newy;
        newy -= step;
        if (newy >= y6)
            {
#ifdef DEBUG
            printf("%.2f %.2f %.2f\n",newx,newy,z5);
#endif
            fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z5);
            }
        }
    newy = lastyvalue;
    miss = newy - y6;
#ifdef DEBUG
    printf("miss5 %.2f\n",miss);
#endif

    start_rad = 0;
    if (miss > 0)
        {
        start_rad = miss/rad3;
```

```
                calx = rad3 * cos(start_rad);
                caly = rad3 * sin(start_rad);
                newx = x6 - (rad3 - calx);
                newy = y6 - caly;
#ifdef DEBUG
                printf("%.2f %.2f %.2f\n",newx,newy,z5);
#endif
                fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z5);
                }

        perstep_rad = step/rad3;
        for (i = 0; newx >= x7; ++i)
                {
                start_rad += perstep_rad;
                calx = rad3 * cos(start_rad);
                caly = rad3 * sin(start_rad);
                lastxvalue = newx;
                lastyvalue = newy;
                newy = y6 - caly;
                newx = x6 - (rad3 - calx);
                if (newx >= x7)
                        {
#ifdef DEBUG
                        printf("%.2f %.2f %.2f\n",newx,newy,z5);
#endif
                        fprintf(fp,"%.2f %.2f %.2f\n",newx,newy,z5);
                        }
                }

        newx = lastxvalue;
        newy = lastyvalue;
        miss = newx - x7;
#ifdef DEBUG
        printf("miss6 %.2f\n",miss);
#endif

        if (miss > 0)
                {
                newx = x7 - miss;
#ifdef DEBUG
                printf("%.2f %.2f %.2f\n",newx,y8,z8);
#endif
                fprintf(fp,"%.2f %.2f %.2f\n",newx,y8,z8);
                }

        for (i = 0; newx >= x8; ++i)
                {
                lastxvalue = newx;
                newx -= step;
                if (newx >= x8)
                        {
#ifdef DEBUG
```

```
                        printf("%.2f %.2f %.2f\n",newx,y8,z8);
#endif
                        fprintf(fp,"%.2f %.2f %.2f\n",newx,y8,z8);
                        }
                }
        newx = lastxvalue;
        miss = newx - x8;
#ifdef DEBUG
        printf("miss7 %.2f\n",miss);
#endif
        fclose(fp);
}   /* main */
```

J. ROAD.H

```
typedef float Dimension;

#include "gl.h"
#include "device.h"
#include "math.h"
#include "time.h"
#include "stdio.h"

#define SYSTEM_ORDER 4
#define MOUNTAIN     8
#define MOUNTAIN1    9
#define SKY          10
#define FIELD        11
#define WARN         12

#define WALL         13
#define SIDEWALL     14
#define ROOF         15
#define WINDOW       16
#define CHMWALL1     17
#define CHMWALL2     18
#define SIDEROOF     19
#define FRAME        20
#define SIDEWALL1    21
#define WALL1        22
#define ROOF1        23
#define FRAME1       24
#define WINDOW1      25

#define MAXFUEL      3000.0
#define PI           3.14
```

K. MAKEFILE

```
CFLAGS = -Zf
SRCS = other.c \
     integrate.c \
     display.c \
     letter.c \
     help.c \
     find_subgoal.c \
     circuit.c \
      carsimu.c

 OBJS = other.o \
      integrate.o \
      display.o \
      help.o \
      carsimu.o \
      find_subgoal.o \
      circuit.o \
      letter.o

 carsimu: $(OBJS)
      cc -o carsimu $(OBJS) -Zf -Zg -lm

 $(OBJS): road.h
```

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
Cameron Station
Alexandria, Virginia 22304-6145

2. Library, Code 0142 2
Naval Postgraduate School
Monterey, California 93943-5002

3. Center for Naval Analyses 1
2000 N. Beauregard Street,
Alexandria, VA 22311

4. Director of Research Administration 1
Code 012
Naval Postgraduate School
Monterey, CA 93943

5. Mr. Russell Davis 2
HQ, USACDEC
Attention: ATEC-IM
Fort Ord, California 93941

6. LCDR H. R. Everett 1
Naval Ocean Systems Center, Code 442
San Diego, California 92152

7. Dr. William E. Isler 1
Defense Advanced Research Projects Agency / ISTO
1400 Wilson Blvd.
Arlington, VA 22209

8. Dr. Andrew Chang 1
Central Engineering Laboratories
FMC Corporation
1185 Coleman Ave, Box 580
Santa Clara, CA 95052

9. Dr. James Lowrie 1
Mail Stop T0427
Martin Marietta Denver Aerospace
P.O. Box 179
Denver, Colo. 80201

10. Dr. D. Y. Tseng 1
Artificial Intelligence Center
Hughes Research Laboratories
23901 Calabasas Rd.
Calabasas, CA 91302-1579

11. Prof. R. E. Fenton 1
Dept. of Electrical Engineering
Ohio State University
2015 Neil Ave.
Columbus, OH 43210

12. Prof. K. W. Olson 1
Dept. of Electrical Engineering
Ohio State University
2015 Neil Ave.
Columbus, OH 43210

13. Dr. Robert B. McGhee, Code 52Mz 20
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943-5000

14. Dr. Michael J. Zyda, Code 52Zk 80
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943-5000