A System for Fast Navigation of Autonomous Vehicles

Sanjiv Singh, Dai Feng, Paul Keller, Gary Shaffer, Wen Fan Shi,
Dong Hun Shin, Jay West, and Bao Xin Wu

CMU-RI-TR-91-20 2

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

September 1991

Table of Contents

List of Figures

Abstract

This report describes an autonomous mobile robot designed to navigate at high speeds over featureless terrain– terrain which is not navigable by relying on features that are found on paved roads and highways. To this end we have developed a paradigm that we call *position based navigation* that relies on explicit vehicle position information from inertial and satellite instruments, to navigate. Specifically we have tackled four main areas- path tracking, which guides the robot vehicle over a pre-specified path, obstacle detection, which is responsible for bringing the vehicle to a stop from high speed when an obstacle is detected, obstacle avoidance, which is responsible for steering the vehicle around detected obstacles so as to rejoin the specified path, and, computing architecture, which integrates all the capabilities into one system. We discuss the algorithms and the devices that were implemented on NavLab, a navigation testbed at Carnegie Mellon. The most notable of our results is the 11m/s speed achieved by the vehicle.

1. Introduction

The drive to extend robotics from factory environments into the natural outdoors has been slow, in part because traditional mobile robots have relied heavily on environmental structure navigate. An automated factory vehicle might be guided by wires buried under the floor, by painted lines, or by some other form of markers. Operating speeds are typically low for such robots and simple, geometric vehicle models suffice for purposes of control. In contrast, little structure can be imposed upon outdoor environments such as those found in a strip mine or a hazardous waste disposal site. Automated vehicles operating there must travel at high speeds and it is no longer possible to represent them with simple models. With outdoor environments in mind, we have devised *FastNav*, a system that provides autonomous navigation for a vehicle equipped with speed and steering control. This report discusses the methods developed and evaluated experimentally over the period May 87- September 89.

In a typical scenario, a reference path for an autonomous vehicle is recorded while a human drives along the intended route. The autonomous vehicle navigates along such paths at high speeds until it detects an obstacle that would interfere with its travel. The vehicle immediately comes to a stop and then plans a path around the obstacle to converge back on to the prespecified route. At this point the vehicle resumes its normal travel to its destination.

To accomplish such a capability for an autonomous vehicle, we have concentrated on four areas:

- *Path Tracking* uses explicit position information from inertial and satellite navigation instruments to provide servo control of steering and speed to keep the vehicle on a specified path. Steering motions are generated by a hybrid control scheme- a feedback scheme attempts to reduce tracking errors while a feedforward scheme compensates for the vehicle's dynamics before the errors build up.
- *Obstacle Detection* uses the fact that the vehicle must follow a specified path within a given tolerance. A section of the path immediately ahead of the vehicle is searched for obstacles using range data from a scanning laser rangefinder. Several schemes have been tried and we have had the most success with one in which an expectation of the road profile is built up as the vehicle traverses its path. Gradual changes in the road profile are progressively integrated into the expectation and abrupt changes (due to obstacles) can be detected easily.
- *Obstacle Avoidance* leads the vehicle around obstacle(s) to the prespecified path. In this mode, the steering angle is calculated repeatedly based on range data from visible faces of obstacles.
- The fourth main area is the development of a *real-time computing architecture* to allow the above mentioned capabilities to function concurrently and cooperatively in an integrated system.

As with any other large project we had to develop our own software utilities and build

some our own devices. Two efforts stand out. The first is range data simulator that simulated range images from a range scanner mounted on a vehicle, moving around in a terrain that can be specified by a user. The simulated range images were used widely in our testing of obstacle detection and obstacle avoidance. The other is a laser range scanner, *Cyclone*, designed and fabricated to provide range data in our navigation experiments.

We have implemented our schemes on NavLab (Figure 1), a navigation testbed that was designed and built at Carnegie Mellon[10]. NavLab is a modified two-ton Chevy van equipped with regulators for steering and velocity control that can be set by application of an analog voltage. These controls are manipulated by host computers on-board to guide the vehicle based on a variety of sensing modes that are available. Our project has used vehicle positioning from inertial instruments, and a scanning laser rangefinder to navigate.

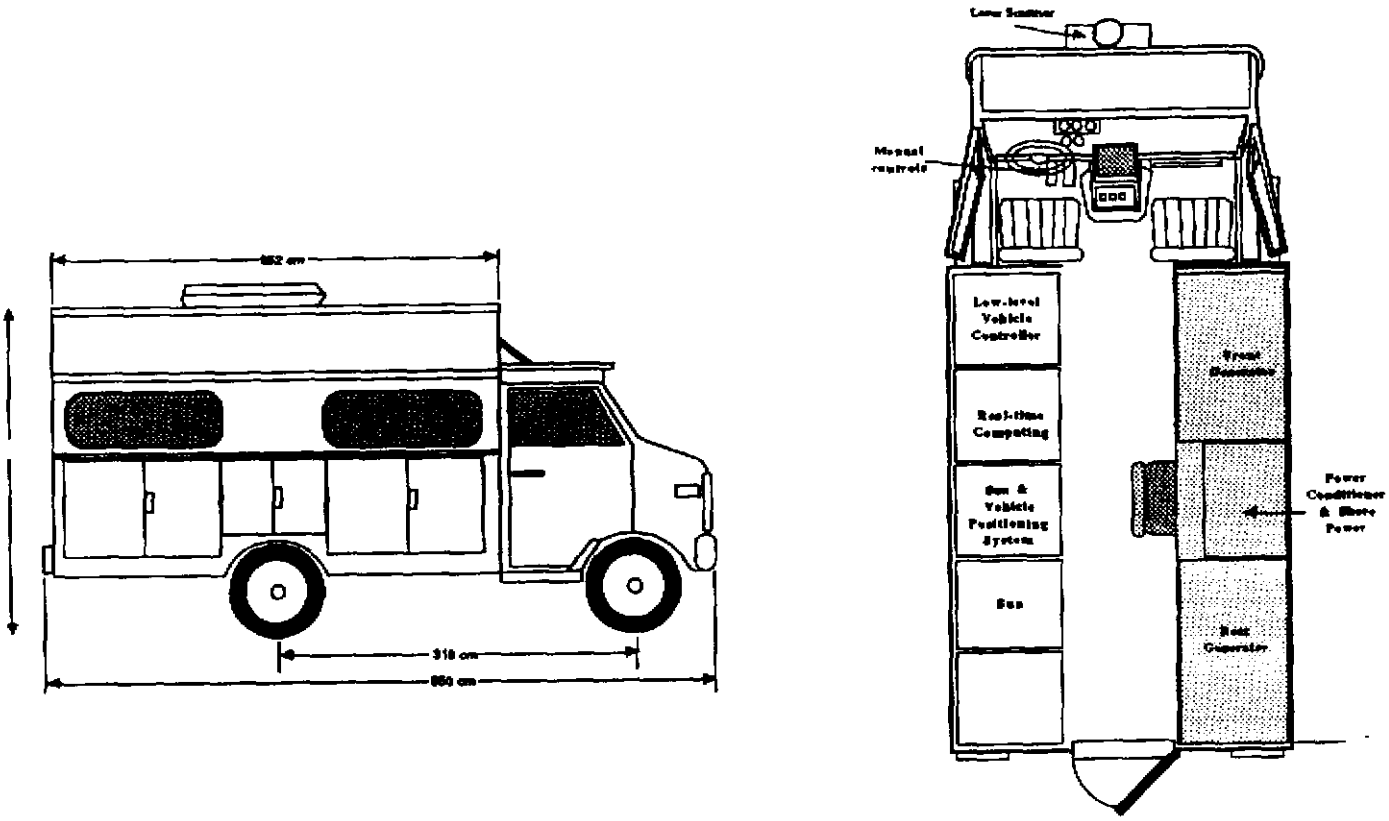

Figure 1: NavLab, an autonomous navigation testbed

Section 2 discusses Path Tracking in detail. Vehicle and path models are developed and the effect of intrinsically "good" paths on tracking performance is shown. A feedback scheme based on the geometry of tracking errors is proposed and its limitations are presented. A feed-forward scheme that attempts to compensate for vehicle dynamics is presented and the improved results are shown.

Section 3 discusses several methods of detecting obstacles. First a two-dimensional scheme is presented that uses range profiles in a horizontal plane to detect obstacles. The limitations of this scheme are discussed and two other schemes are presented that extend obstacle detection into a three-dimensional world. The first selects only a few lines of

range data to detect obstacles in front of the vehicle. The other scheme uses the entire range image to find obstacles. The main idea in both 3-D schemes is to fit range data to an incrementally updated road model. Deviations from the road model are considered to be obstacles.

Section 4 discusses an obstacle avoidance scheme that uses a two-step strategy to plan a path around the obstacles in the path of the vehicle. First, a global scheme picks a "subgoal" based on its current state and the final destination of the vehicle. A local scheme is used to make steering decisions in a low-level control loop based on a potential function.

Section 5 discusses the computing architecture that was developed specifically to integrate the various parts of the system developed. We propose an architecture in which it is possible to build a distributed system running on several processors. Some design principles that were distilled from our experience are discussed.

Section 6 discusses the incorporation of an array processor into our computing architecture to tackle computing bottlenecks. Benchmarks comparisons of two commonly used applications with and without the array processor are presented.

Section 7 discusses a range simulator that provides the capability of modeling a terrain with obstacles and allows for simulation of range images obtained from a range scanner moving around in the modeled terrain.

Section 8 discusses two devices that were designed and fabricated for the project. Cyclone is a scanning laser rangefinder that was used for obstacle detection and obstacle avoidance. NavArm is a large rotating arm that was used to test obstacle detection algorithms.

Section 9 highlights the final results achieved and summarizes the experience we have gained from this project.

2. Path Tracking

We propose a systematic methodology for high performance path tracking by full sized robot vehicles. The operation of such vehicles is distinguished from the operation of their factory counterparts in two major ways. Firstly, these vehicles cannot rely on structure in the environment to help in navigation- no painted lines or guide rails exist in vast open areas that such vehicles must often negotiate. Further, their behavior cannot be represented by simple kinematic models- schemes that consider the robot as a massless geometrical entity do not succeed at high speeds. We have dealt with both these issues and discuss them in detail in this section. We have assumed that the robot is a full-sized conventionally steered vehicle, such as a car or truck. Thus, it has two actuation trains: steering and propulsion. We also assume that the robot can sense its global position, orientation and velocity.

The methodology was first developed and implemented using computer simulation. The techniques were then calibrated and refined by experimentation on NavLab at speeds up to 35 km per hour. In this section we will discuss kinematic and dynamic vehicle models, the proposed path tracking control scheme and experimental results.

2.1. Vehicle Model

Kinematic vehicle models are based purely on the geometry of the vehicle. They describe, to a first order of approximation the behavior of a vehicle given its dimensions, and a time history of its speed and steering angle. Dynamical models on the other hand capture such considerations as mass, friction, and the interaction between the vehicle and the ground.

2.1.1. The Choice of a Guide Point

For our analysis, we use a bicycle model as an archetype of conventionally steered vehicles. The guide point of the vehicle, which is guided along the reference path, is chosen as the midpoint of the rear axle, as in Figure 2. This choice is fortuitous for several reasons:

- The steering angle at any point on the path is determined geometrically, independent of the speed, in the following manner:

$$\tan\phi = \frac{l}{r} \tag{1}$$

  where l is the wheelbase of the vehicle and c is the curvature of the path. Also, the angular velocity of the driving rear wheel is determined only by the vehicle speed, v, and the radius of the wheel, R_w:

$$w = \frac{v}{R_w} \tag{2}$$

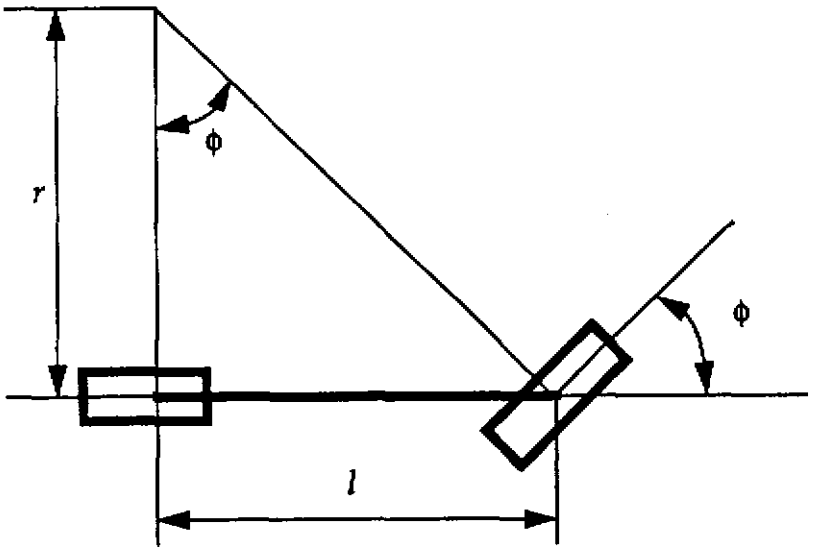

Figure 2: Geometry of a bicycle

If the guide point is placed elsewhere, expressions for ϕ and ω are more complex than (1) and (2). The reference steering angle and angular velocity of the driving wheel must be obtained by numerical integration.

- The vehicle follows the minimum turning radius for the maximum steering angle [20]. In other words, the peak steering angle is smaller than for any other choice of guide point.
- The heading of the vehicle is aligned with the tangent direction of the path. This gives a more reasonable vantage point for a vision camera or a range scanner mounted at the front of the vehicle, as in Figure 3, and a smaller area is swept by the vehicle.

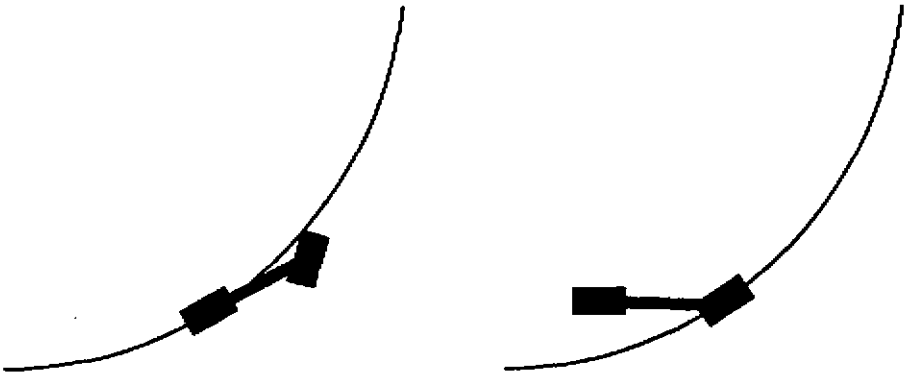

Figure 3: Heading alignment along a path

For the guide point at the center of the rear axle, certain paths will require infinite accel-

eration of the steering wheel. This is not the case if the guide point is moved from the rear axle, for example, to the front wheel. However, this choice loses the advantages discussed above.

We define "posture" as the quadruple of parameters (x, y, θ, c) which describes the state of a conventionally steered vehicle. A sequence of postures are used as a path in our path tracking.

2.1.2. Vehicle Kinematics

In this subsection, we discuss vehicle kinematics of the vehicle model, that is the motion of the vehicle without regard to the forces that cause it. We show the relation between the control actions on the vehicle (steering and propulsion) and the resulting changes in position and orientation with respect to a world coordinate frame.

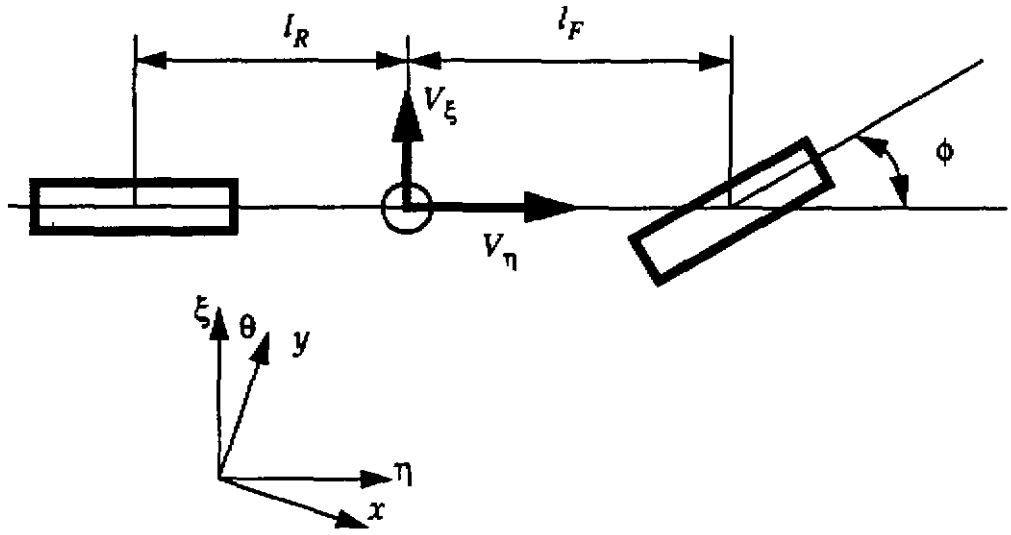

Figure 4: Bicycle model and the corresponding coordinate systems

The bicycle model kinematic coordinate systems are assigned as shown in Figure 4. The body coordinate system (η, ξ) has its origin at a reference point of the vehicle, for example, the guide point or the mass center, and it rotates at the same angular velocity as the vehicle body. This simplifies the description of the motion of a vehicle because there is no motion of the mass point with respect to the body coordinate system in the world. The global coordinates, x and y, describe the motion of the vehicle system; v_η and v_ξ are global velocities. Note that x and y are also the inertial coordinates, since the coordinate frame does not move. The local coordinates represent the motion of a part of the vehicle, for example, ϕ, ω_R for the motions of the steering and the wheels. The following kinematic equations between (ϕ, ω_R) and (v_η, v_ξ, θ) result:

$$\dot{\theta} = \frac{R_W \tan\phi}{l_F + l_R} R_W \omega_R \tag{3}$$

$$v_\eta = R_W \omega_R \tag{4}$$

$$v_\xi = \frac{l_R R_W \tan\phi}{l_F + l_R} \omega_R \tag{5}$$

where w_R is an angular speed of the rear wheel and R_w is the radius of the wheel. Note that v_h and v_x have no corresponding coordinates because they are expressed in terms of a rotating body coordinate system (η, ξ). These velocities are referred to as quasi-coordinates velocities. In many cases, they are more useful than generalized coordinate velocities in describing the motion. To compute the vehicle position from the local coordinate system values f and w_R, equations (4) and (5) are converted into the following equations between local coordinates f w_R and global inertial coordinates x, y.

$$\dot{x} = \left[\cos\theta - \frac{l_R \tan\phi}{l_F + l_R} \sin\theta\right] R_W \omega_R \tag{6}$$

$$\dot{y} = \left[\sin\theta - \frac{l_R \tan\phi}{l_F + l_R} \cos\theta\right] R_W \omega_R \tag{7}$$

and then the vehicle speed is

$$v = \sqrt{1 + \left(\frac{l_R \tan\phi}{l_F + l_R}\right)^2} \, (R_W \omega_R) \tag{8}$$

and the kinematics between (x, y, θ) and (v_η, v_ξ, θ) are

$$\dot{x} = v_\eta \cos\theta - v_\xi \cos\theta \tag{9}$$

$$\dot{y} = v_\eta \sin\theta + v_\xi \cos\theta \tag{10}$$

These forward kinematic equations are applied to generate global coordinate values (x, y, θ) or (v_η,v_x, θ) from local coordinate values (ϕ, ω_R). Note that the forward kinematic equations are non-exact differential equations because of the existence of quasi-coordinates. This fact will affect the control scheme for path tracking. For example, control schemes

based on actuator, coordinate-like, joint-based schemes for manipulators cannot be used for path tracking by robot vehicles, because even perfect servo-control of the actuators cannot compensate for vehicle position errors.

2.1.3. Vehicle Dynamics

It is common in the control of autonomous vehicles to establish the necessary kinematic models but to ignore an explicit representation of the vehicle dynamics or, alternatively, to compensate for them by using "fudge factors". It is important to model the dynamics of the vehicle for the following two reasons:

- Since vehicle actuators have less than perfect response, vehicle behavior differs from the kinematically computed behavior; a good dynamic model can be used to compensate for this discrepancy.
- A good model of the vehicle is required for simulation, so that algorithms can be tested without having to experiment directly on the vehicle.

The dynamics of the vehicle describe the relationship between the inputs to the actuators and the behavior of the vehicle. In our analysis, the dynamic model for robot vehicles is separated into two parts: **chassis dynamics**, and **vehicle ground interaction** (VGI), as in Figure 5.

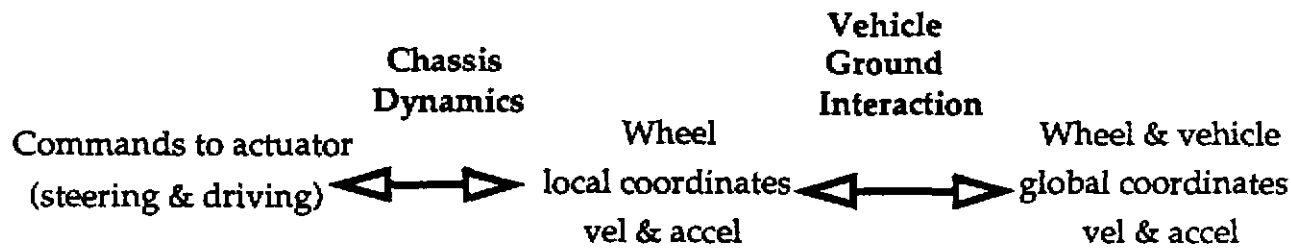

Figure 5: Partitioned vehicle dynamics

Chassis dynamics describe angular velocities and angular accelerations of vehicle wheels, given inputs to the vehicle actuators. VGI is a phenomenon specific to wheeled vehicles and describes the motion of the wheel and the vehicle, given steered wheel angles ϕ and wheel angular speed ω_R.

Chassis dynamics are modeled as two (steering and propulsion), decoupled, first-order systems for both control and simulation. For example, the steering dynamics can be described by:

$$\frac{\phi_{actual}(s)}{\phi_{desired}(s)} = \frac{1}{1+Ts} \tag{11}$$

where T is the time constant of the system model. The step input response of such a system is given by:

$$\phi_{actual}(t) = (\phi_{desired} - \phi_{initial})\,[1 - e^{(-t)/T}] \tag{12}$$

This model is quite simple compared to typical dynamic models used for robot manipulators. The two actuation systems have weak coupling and can be considered effectively decoupled. Both steering and propulsion enable several mechanical systems, and each system has compliance and nonlinearity, hence, in practice it is intractable to model the dynamics of all mechanical parts and to identify their parameters.

We have examined a higher order model to account for some of the nonlinearities in vehicle motion. These higher order models are used in simulating vehicle motion but not for purposes of planning steering motions. This model attempt to describe the vehicle-ground interaction (VGI). More specifically it describes the motion of the vehicle given steered wheel angles, wheel angular speeds, vehicle mass, friction between the ground and the wheels, and tire stiffness.

Several researchers [5, 6, 7, 22] have shown that the principal VGI effects are caused mainly by the distortion of the tires. In general, VGI is specified as the amount of *slip angle* for various amounts of lateral force on the tire: slip angle is the angle between the centerline of the tire carcass and the velocity vector of the wheel. We have used the following linear approximation:

$$F_{lateral} = c_\alpha \alpha \tag{13}$$

Where $F_{lateral}$ is the lateral force to the wheel from the ground, c_α is the cornering stiffness, and α is the slip angle. This linear model is known to be valid at the lateral accelerations up to 3 m/s^2 [5].

2.1.4. Systemic Effects

Often, it is easy to ignore systemic effects, such as communication and computing delays, in the modeling of a vehicle, especially because they are not visible at low tracking speeds. There are several delays in the implementation of the path tracking control scheme that are caused by finite I/O bandwidth and computing loops. Timing of control commands is another concern at high speeds. With an operating system like Unix, the procedure is to poll a clock to ensure that commands are sent at regular intervals. However, such operating systems do not guarantee real-time response, and the interval between commands can vary up to 0.5 second on the NavLab. Our dynamic model takes these systemic effects into account.

2.2. Path Tracking Algorithm

The path tracking problem is approached by first formulating a control problem that addresses the main characteristics of conventionally steered vehicles: non-holonomic constraints and kinematically decoupled propulsion and steering. A scheme is presented to specialize tracking control that is applicable to full-sized roadworthy vehicles. These vehicles defy complete, precise modeling due to difficulty in modeling complex subsystems. The scheme circumvents the complete modeling, yet provides good perfor-

mance of tracking by feedforward compensation for latency and by smooth compensation for instantaneous errors. Latency appears to be the dominant characteristic in the behavior of a slow-responding system.

2.2.1. Problem Formulation

The kinematic bicycle model reveals two main characteristics that impact path tracking control:

- These non-holonomic systems are constrained by the non-algebraic differential equation describing the relationship between the steering angle and vehicle position. Consequently, the vehicle position error cannot be compensated only by steering feedback control. Thus, the control scheme should be based instead on the global description of vehicle position. However, the traditional sampling rate for sensing global positions is very slow, whether by kinematics or by direct sensing, and causes the resulting closed-loop system to succeed only at a low frequency. This degrades the stability of the system, as well as its capability to reject disturbance.
- The center of the rear axle is selected for the position of the guide point. Consequently, steering and propulsion are kinematically decoupled. In other words, geometric path following relies only on steering. Vehicle speed is controlled purely by propulsion.[1]

In order to overcome the former and exploit the latter, the following three strategies are used:

(1) *Path tracking*: we formulate the problem as "path" tracking rather than trajectory tracking. In the typical manipulator tracking problem, a trajectory (time history of position) is planned off-line from a given geometric path, and is then followed in real time. For mobile robots, path tracking is appropriate for the following reasons:

- When a human drives a car along a road, he or she cares most about keeping a car within the road boundaries. Vehicle speed is not changed to follow a time history of vehicle positions. Speed is changed in response to some higher level characteristics such as road curvature, proximity to obstacles, or conditions of the road, vehicle, and the human driver.
- We are more concerned with lateral errors from a reference path than with longitudinal errors along the path. In contrast, trajectory tracking must concern itself with the coupling between both types of error and will, therefore, be more difficult to accomplish. For example, a speed error, such as loss of traction from a disturbance, will result in accelerated propulsion and jerky steering motions in order to recover.
- Most roadworthy vehicles have limited acceleration that is produced

1. Steering and propulsion are weakly coupled dynamically, as mention earlier but the practical implication is only that slightly different torque is required to obtain the desired steering motion, depending o propulsion.

through a multi-step transmission. However, trajectory tracking requires frequent, responsive changes of vehicle speed to compensate for longitudinal error, which is not desirable for tracking a roadworthy vehicle.

- Trajectory planning is usually a separate task; path tracking is less complex and can be performed 'on-line'.

(2) *Reduction to steering problem:* By locating the guide point on the center of the rear axle the reference path directly suggests a control law for steering independent of vehicle speed. Hence, the path tracking problem is then reduced to steering along a given path. Speed is a separate parameter, and is set at a higher level by criteria such as maximum allowable lateral acceleration and proximity to obstacles.

(3) *Nested feedback control loops*: When a path to be tracked is specified by a series of points given in global coordinates, the control of a vehicle should be based not only on actuator coordinates but also on feedback of global position. If closed loop control of steering and propulsion is used alone (without global position and heading feedback information as supplied by the outer feedback loop in Figure 6) in an attempt to follow this pre-specified path, vehicle position and heading errors will accumulate. This is because position and heading result from integrating the entire time history of steering and propulsion.

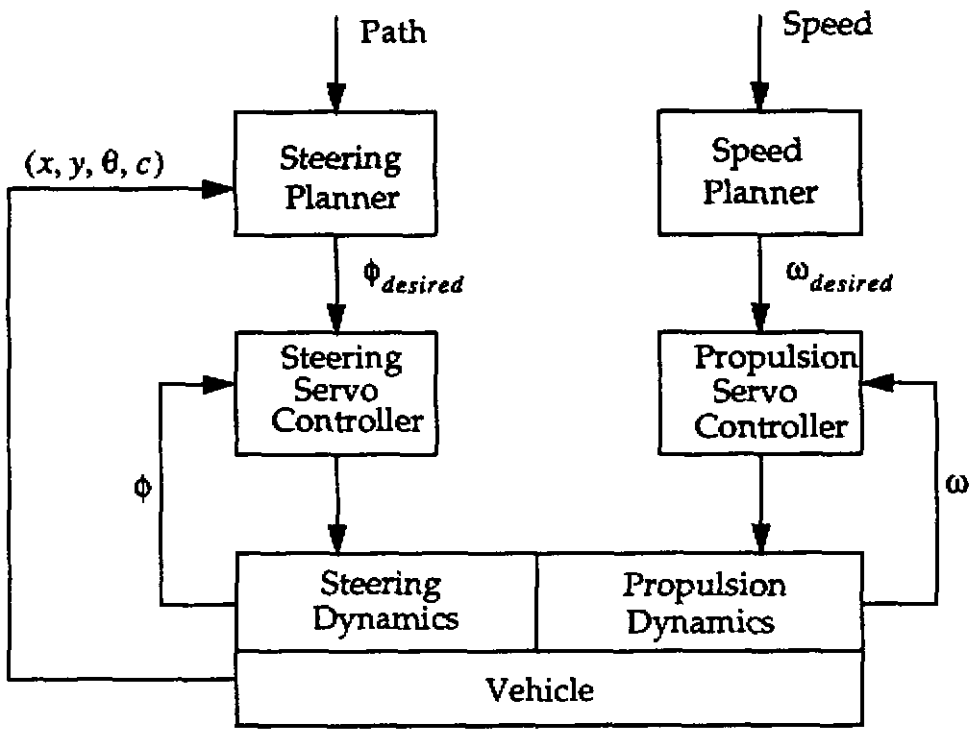

Figure 6: Nested feedback control loops

For example, imagine that a step input is given to the steering system. After the transient response has subsided, the steering mechanism will have reached the desired steering angle. However, in failing to track the step input exactly, offset errors have been introduced in the resultant path, which will have the required

curvature but will be parallel to the reference path (Figure 7).

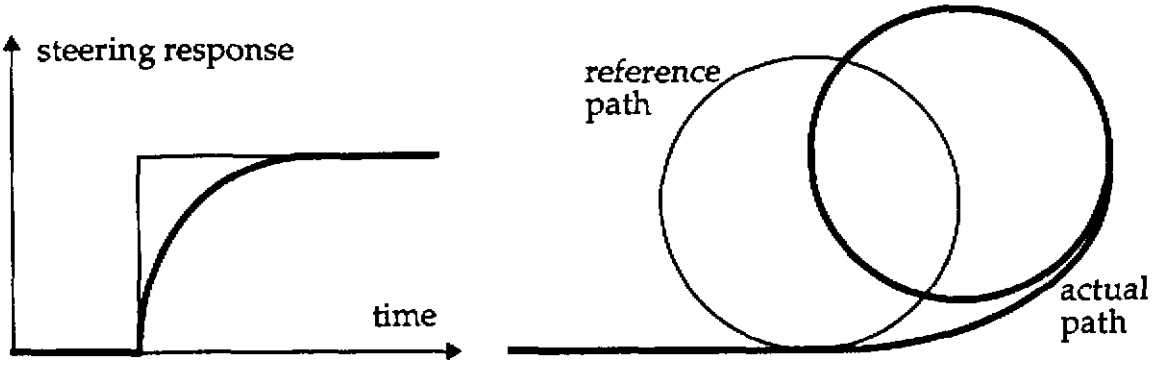

Figure 7: Steering response to a step input and resulting path

Alternately, closed-loop control, based only on global coordinates, is not sufficient, either. In this case, actuators would be servo-controlled based on global errors, as are global control schemes for manipulators. The resulting system would succeed only at low frequency, because vehicle position estimates are only possible at slow rates, either by dead reckoning or direct position sensing. This would, in general, degrade the stability and disturbance-rejection capabilities of the system. Consequently, it is necessary to feedback global vehicle position and heading, in addition to servo-control feedback.

The advantages of geometric path tracking, a nested feedback loop, and kinematically decoupled steering and propulsion reduce the path tracking problem to a tractable steering planning control problem and actuator servo control. In other words, for a mobile robot to track paths satisfactorily, it is necessary to generate steering and propulsion commands for the servo-controllers with global position feedback and to execute these commands at the real time servo level. Since servo-control is already a mature technology and steering decides how well the path is tracked, this research has emphasized real-time steering command generation which we call *steering planning*.

2.2.2. Preview Control and Partitioned Schemes

In general, tracking control, the problem of generating the control inputs, U_i, has both feed forward and feedback compensation:

$$U_i = R_i + Ke_i \tag{14}$$

R_i denotes feedforward compensation, e_i is the error between the reference input and actual state of the system, and K is a gain. The $K.e_i$ term denotes an error feedback compensation; for example, it can be any type of Proportional-Integral-Derivative (PID) action. A feedforward compensation is typically the compensation for expected disturbances to the system or open-loop control through the computation of the inverse dynamics.

The typical way to solve the tracking problem is to formulate it as a regulator problem and to change its references as the desired input states change. Then, the method is basi-

cally a feedback scheme which stabilizes *bounded input and bounded output*. However, a vehicle driving at high speeds requires more than this type of control in order to overcome the limited rate of response of the system dynamics. *Preview control* is introduced as an additional control scheme to improve vehicle tracking ability. The term *preview* implies that the future desired states of the commands are known in advance. Human vehicle driving is known as a typical example of preview control [27, 31, 6].

One characteristic of preview control is that *a priori* knowledge of the future inputs can be used to improve performance. Intuitively, it is possible that some of the system errors are fortuitous in light of the future reference path, and thus can be exploited. Figure 8 contrasts a conventional feedback scheme with a scheme that uses a "preview" of future inputs. In Figure 8, at time t, the controller uses knowledge of the desired states in the interval $[t \rightarrow t + l_a]$ where l_a denotes the preview distance, to plan the next control inputs.

Another characteristic of preview control, which is not described in Figure 8, is that anticipation of the system behavior can be used to improve performance. Thus, another way of looking at preview control is that feedback control only compensates for errors after the fact; whereas, knowledge of future desired states and an approximate dynamical model of imperfect system behavior can be used to compensate for errors before the fact.

Thus, a viable controller for high performance tracking of autonomous vehicle should have both feedback and feedforward compensation capabilities (Figure 9). The steering planner is partitioned into two parts:

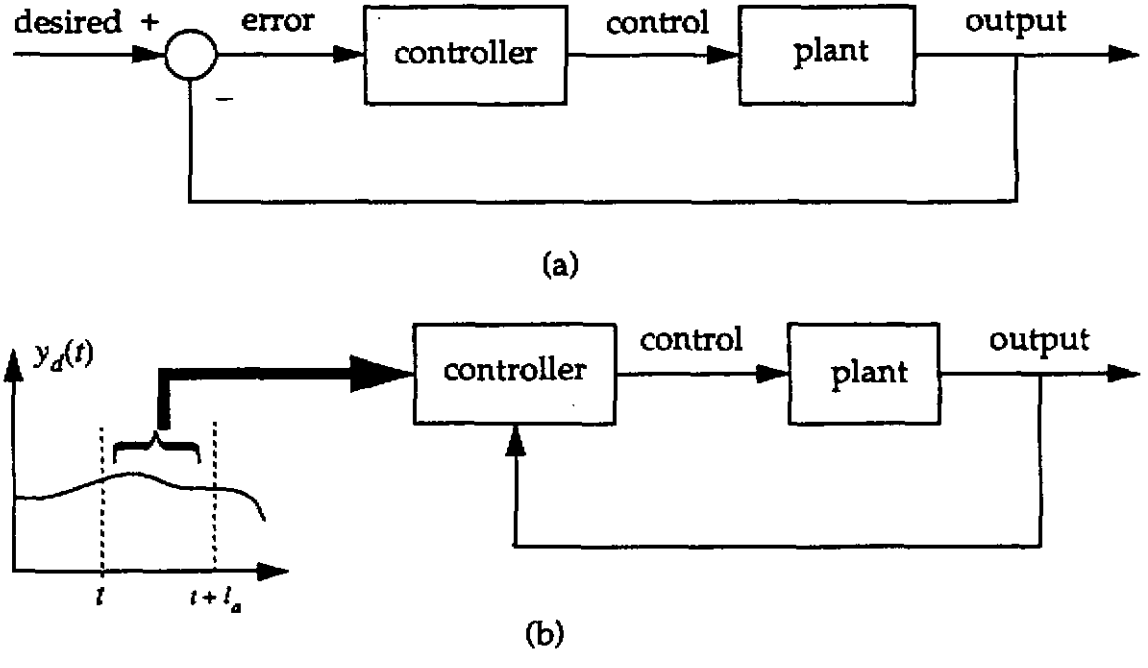

Figure 8: (a) Conventional control (b) Preview Control

- *feedback compensator*, which provides closed-loop compensation for errors between the vehicle's actual path and its desired path;

- *feedforward compensator,* which uses current reference inputs but also incorporates the future course of the path anticipating the behavior of the system.

Partitioning the controller helps to understand the task of the controller more clearly and makes it easier to design and implement.

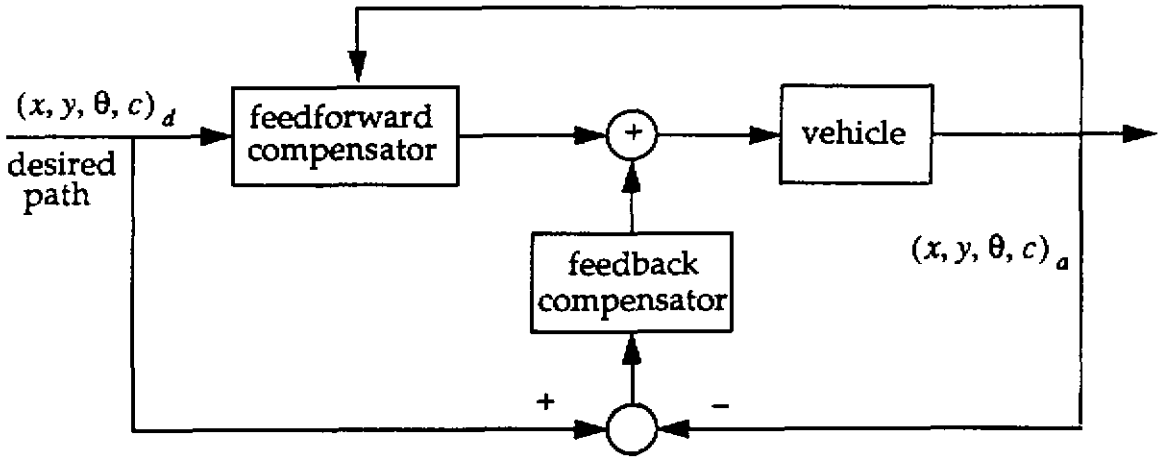

Figure 9: Partitioned scheme using both feedforward and feedback compensation

One way to think about such a hybrid scheme is that the feedback scheme provides correction for the errors after the fact, that is after the errors have resulted, while the feedforward scheme provides adjustment for the errors before they build up.

2.2.3. Feedback Compensator

The feedback compensator guides a vehicle by closed-loop compensation for instantaneous deviation from the prescribed path. The developed scheme replans a simple, continuous path that converges to a desired path in some look-ahead distance. It also computes a steering angle corresponding to the part of the replanned path, which will be followed for the next time interval. Since it does not require explicit dynamic models of the system, this scheme does not guarantee stability of the closed-loop system analytically, but provides a simple convergence of errors to the desired path, especially for a low frequency system, like the steering control of a roadworthy vehicle.

Consider a path as a continuous function ($(x(s), y(s))$ and a vehicle that is currently at P_a. An error vector can be calculated that represents error in the distance transverse to the path. At any given moment, the vehicle will have errors in lateral position, heading, and curvature. The scheme uses the geometry of the errors to obtain a continuous quintic polynomial function that converges to zero errors at some prespecified look-ahead distance. Consider a desired continuous path $(x(s), y(s))$ and a vehicle that is currently at P_a. An error vector can be calculated that represents error in the distance transverse to the path (ε_0) relative to P_0 in heading (β_0), and in curvature (γ_0), as in Figure 10.

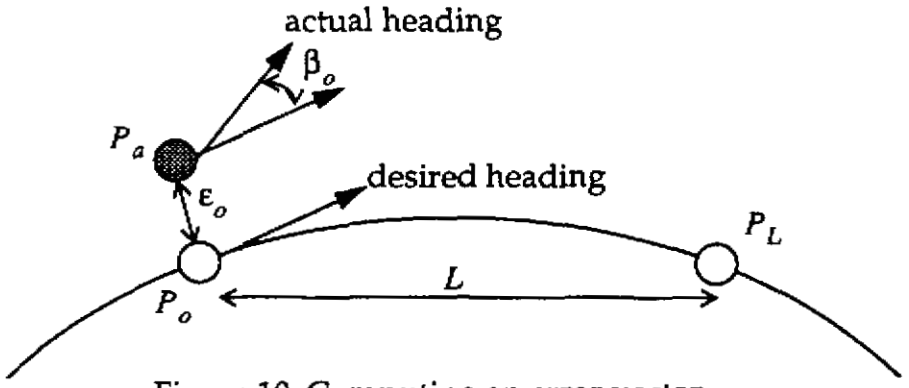

Figure 10: Computing an error vector

If the vehicle is to be brought back onto the specified path within distance L (measured along the reference path), six boundary conditions can be stated corresponding to the initial errors and to zero errors at P_L:

$$\varepsilon(P_o) = \varepsilon_o \quad ; \quad \varepsilon(P_L) = 0$$

$$\beta(P_o) = \beta_o \equiv \left.\frac{d\varepsilon(s)}{ds}\right|_{s=P_o} \quad ; \quad \beta(P_L) = \beta_L \equiv \left.\frac{d\varepsilon(s)}{ds}\right|_{s=P_L} \tag{15}$$

$$\gamma(P_o) = \gamma_o \equiv \left.\frac{d^2\varepsilon(s)}{ds^2}\right|_{s=P_o} \quad ; \quad \gamma(P_L) = \gamma_L \equiv \left.\frac{d^2\varepsilon(s)}{ds^2}\right|_{s=P_L}$$

These boundary conditions can be sufficiently satisfied by a polynomial of fifth degree. A quintic polynomial can be constructed to describe the replanned path (in error space) as follows:

$$\varepsilon(s) = a_0 + a_1 s + a_2 s^2 + a_3 s^3 + a_4 s^4 + a_5 s^5 \tag{16}$$

where $s \in [0, L]$

Solving for $a_0, a_1, a_2, a_3, a_4, a_5$:

$$a_0 = \varepsilon_0 \tag{17}$$

$$a_1 = \beta_0 \tag{18}$$

$$a_2 = 2\gamma_0 \tag{19}$$

$$a_3 = \frac{-(13\gamma_0 L^2 + 12\beta_0 L + 20\varepsilon_0)}{2L^3} \tag{20}$$

$$a_4 = \frac{9\gamma_0 L^2 + 8\beta_0 L + 15\varepsilon_0}{L^4} \tag{21}$$

$$a_5 = \frac{-(7\gamma_0 L^2 + 6\beta_0 L + 12\varepsilon_0)}{2L^5} \tag{22}$$

The expression for $\varepsilon(s)$ gives the error along the path from P_O to P_L. The curvature in error space can be written as:

$$\frac{d^2\varepsilon(s)}{ds^2} = 2a_2 + 6a_3 s + 12a_4 s^2 + 20a_5 s^3 \tag{23}$$

Steering angle variation can be obtained from (23) as follows:

$$\phi_{feedback}(s) = \tan^{-1}\{\frac{d^2\varepsilon(s)}{ds^2} l\} \tag{24}$$

Computing the steering angle $\phi_{feedback}(s)$ over the entire interval $[0 \rightarrow L]$ is not necessary. Instead, during every control loop cycle, the coefficients $a_1 \ldots a_5$ are computed based on the most recent estimate of position, heading, and curvature errors, and $\phi_{feedback}(s)$ is computed only for a small path distance immediately ahead of the vehicle.

The sensitivity of the scheme described above can be modulated by the adjustment of the parameter L which represents the distance at which all three parameter (position, heading, and curvature) errors are forced to be zero. Figure 11 shows the difference in

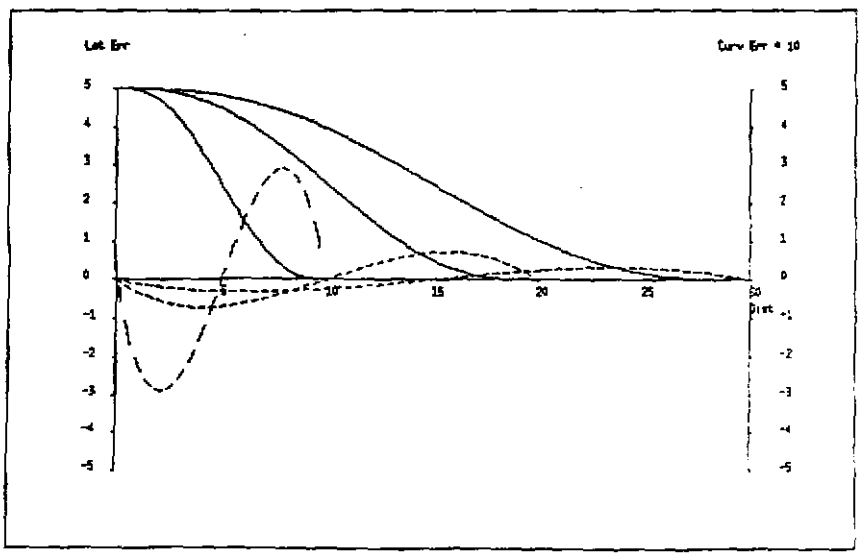

Figure 11: The effect of modulating the parameter L

variation produced by the quintic polynomial for three different values of L. The solid lines denote the variation in position error, while the dashed lines indicate the variation in the curvature.

The parameter L (look-ahead distance) is chosen based on vehicle speed, much as a human driver would choose to look out farther along the path the higher the vehicle speed. A larger value of L corresponds to a milder contribution from the feedback scheme than for smaller values. Once again, a simple linear model is used to vary L with vehicle speed as in (25), the gains for which are determined experimentally:

$$L = slope \bullet (V - V_{ref}) + L_{ref} \tag{25}$$

where V is the speed of a vehicle.

2.2.4. Feedforward Compensator

The dynamic behavior of a vehicle can be anticipated by using an approximate dynamic model of the system. However, the vehicle dynamics are too complicated to model and to compensate for the dynamics exactly. Thus, we have chosen to compensate for the most dominant characteristic behavior of the system dynamics — the latency effect. Since this phenomenon can be described by a time constant in a first-order lag system, steering dynamics are modeled as a first-order lag system as in (11).

The effect of the feedforward compensation can be seen in Figure 12, Figure 13, Figure 14, and Figure 27. Figure 12 shows the response (solid lines) of the steering system to given reference commands (dashed lines). The reference commands correspond to a path composed of a straight line, a circular arc, and a straight line. The reference and resultant paths in this case are shown in Figure 13.

Feedforward compensation is accomplished by sending reference commands in advance. In this manner, the steering starts moving before the arc is reached. A useful amount of advance is the time constant of the first-order model. If the vehicle dynamics are modeled exactly by a first-order lag, then the steering would reach 63% of the desired value by the time the arc path transition is encountered. An intuitive argument can be made that such a scheme produces an area under the responsive curve, (solid line in Figure 14) that approximates the area under the reference curve (dashed line in Figure 14) and at the required time. Figure 27 shows the improvement produced by anticipating the response of the vehicle dynamics.

Note that the future reference steering command is used. Thus the feedforward compensator satisfies both characteristics of preview control. Real vehicle responses are not as nicely predictable as shown, but the above scheme works well especially since reference steering commands have much smaller step changes (due to discrete time control) than those shown in the example.

Another virtue of the feedforward compensation method is robustness against systemic effects, such as pure time delay due to the vehicle's communications between the multi-tasking systems of multi-processors.

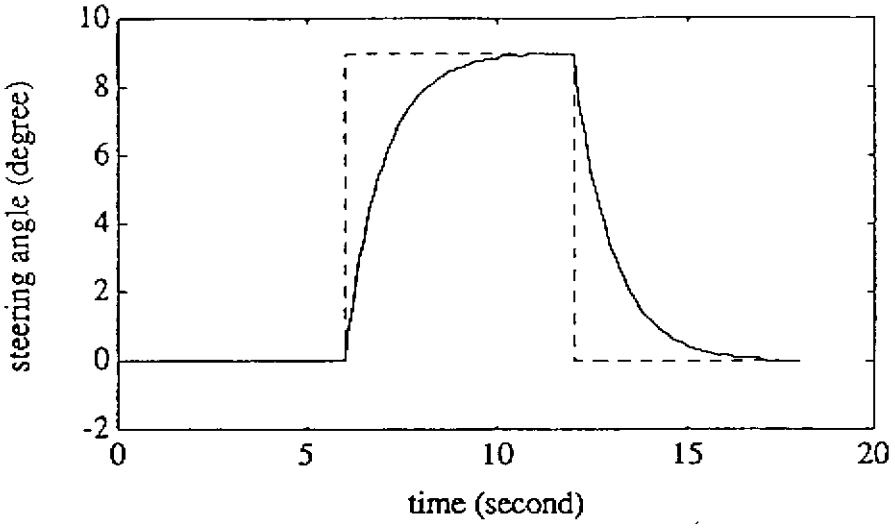

Figure 12: Performance of a first order model of the steering system in response to a path consisting of straight lines and an arc.

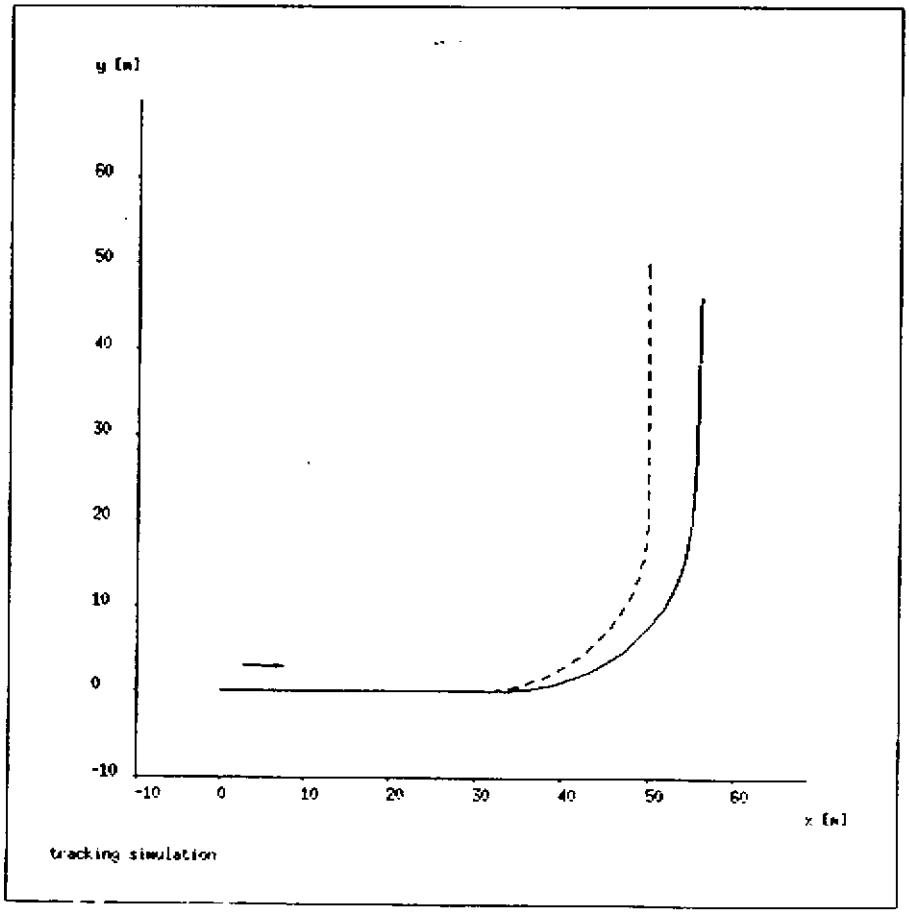

Figure 13: Open-loop response without feedforward compensation

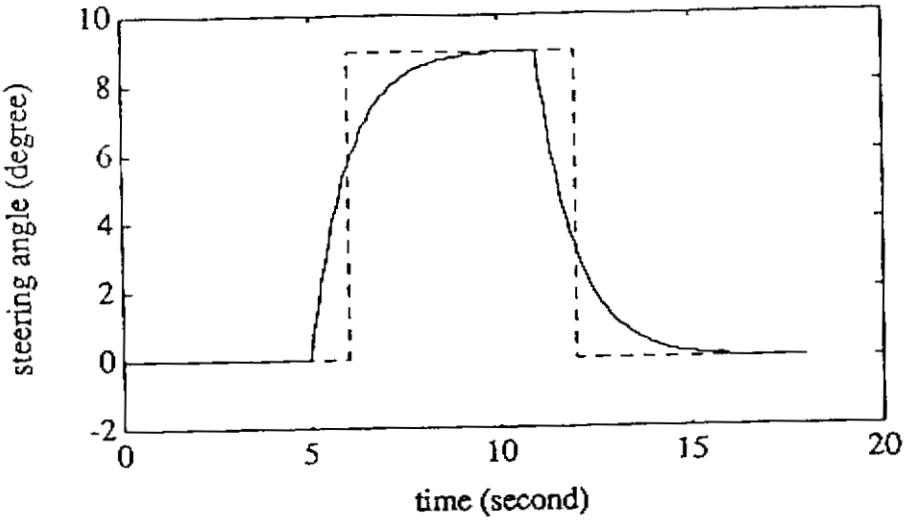

Figure 14: Compensated steering performance modeled as a first order system

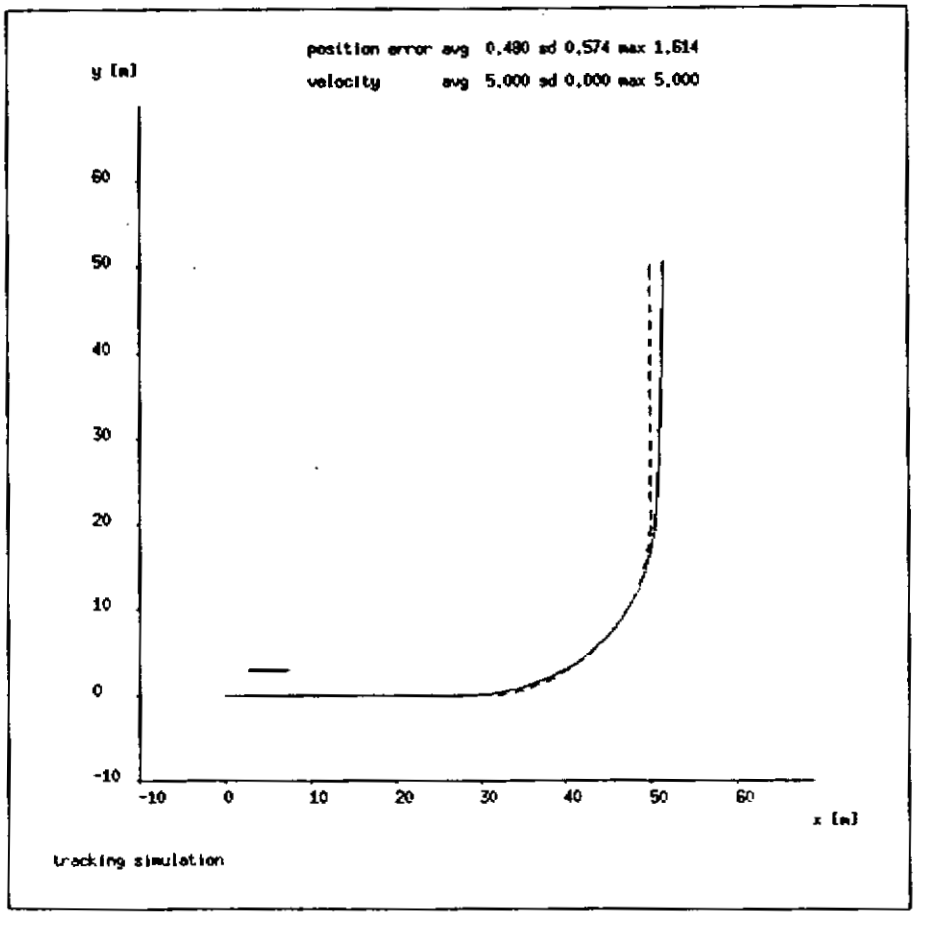

Figure 15: Open-loop response with feedforward compensation

2.2.5. Algorithm Implementation

The path tracking system is closed with nested feedback loops (Section 2.3.1). The inner loop for servo-controllers in the actual vehicle system is executed on the order of 10 mS, while the outer loop is closed normally at the rate of 0.25 second—steering and speed commands are sent to the servo-controllers at this same rate. The time interval for the outer loop is governed by the rate at which the inertial positioning system (section 8.3.) supplies feedback (20 Hz), which is much longer than the computing time required for the Path Tracker (16 mS at SUN 3/75); hence it is necessary for the steering planning to predict the position of the vehicle.

Figure 16 illustrates steering planning procedure in time domain; After sensing the current posture ($P_{a,k}$), the posture at the end of the current time interval ($\tilde{P}_{a,k+1}$) is projected. Then, the desired posture at the end of the next time interval ($P_{d,k+2}$) is computed and the reference steering angle between $\tilde{P}_{a,k+1}$ and $P_{d,k+2}$ is determined.

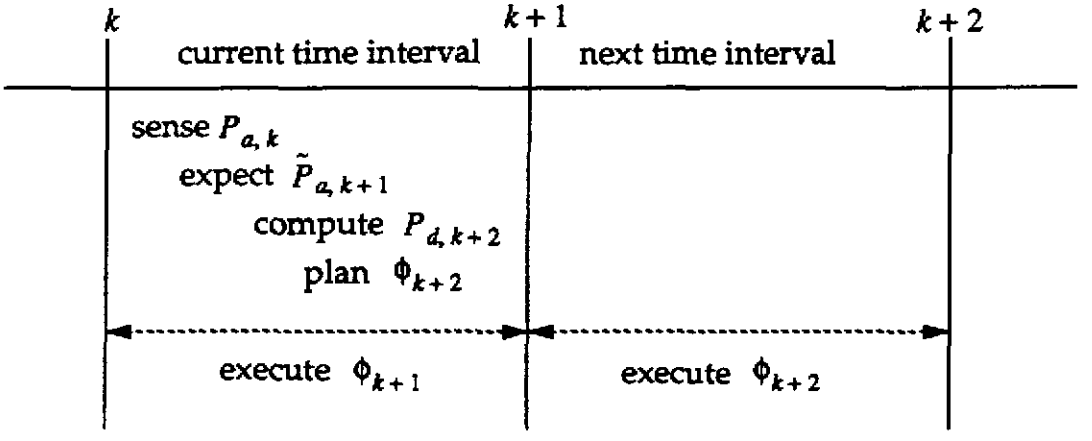

Figure 16: Sequencing steering planning

Ideally, $\tilde{P}_{a,k+1}$ should be computed from $P_{a,k}$, ϕ_{k+1}, the speed of the robot, the planning time interval, and the robot's dynamics. However, because this is too time-consuming, $\tilde{P}_{a,k+1}$ is approximated through the following:

$$\varepsilon_k = P_{a,k} - \tilde{P}_{a,k} \tag{26}$$

$$\tilde{P}_{a,k+1} = P_{d,k+1} + \varepsilon_k \tag{27}$$

The flowchart in Figure 17 illustrates how steering commands are planned in a recursive way, accommodating for the timing and approximation above.

At times, especially when the discrete time interval for steering planning is large, poor prediction of the vehicle position in steering planning degrades performance of the tracking algorithm. A compensation method was developed which reduces the error in predicting the next vehicle position by decreasing the predicting time interval. In this method, a new steering command is planned and then issued as soon as the position of

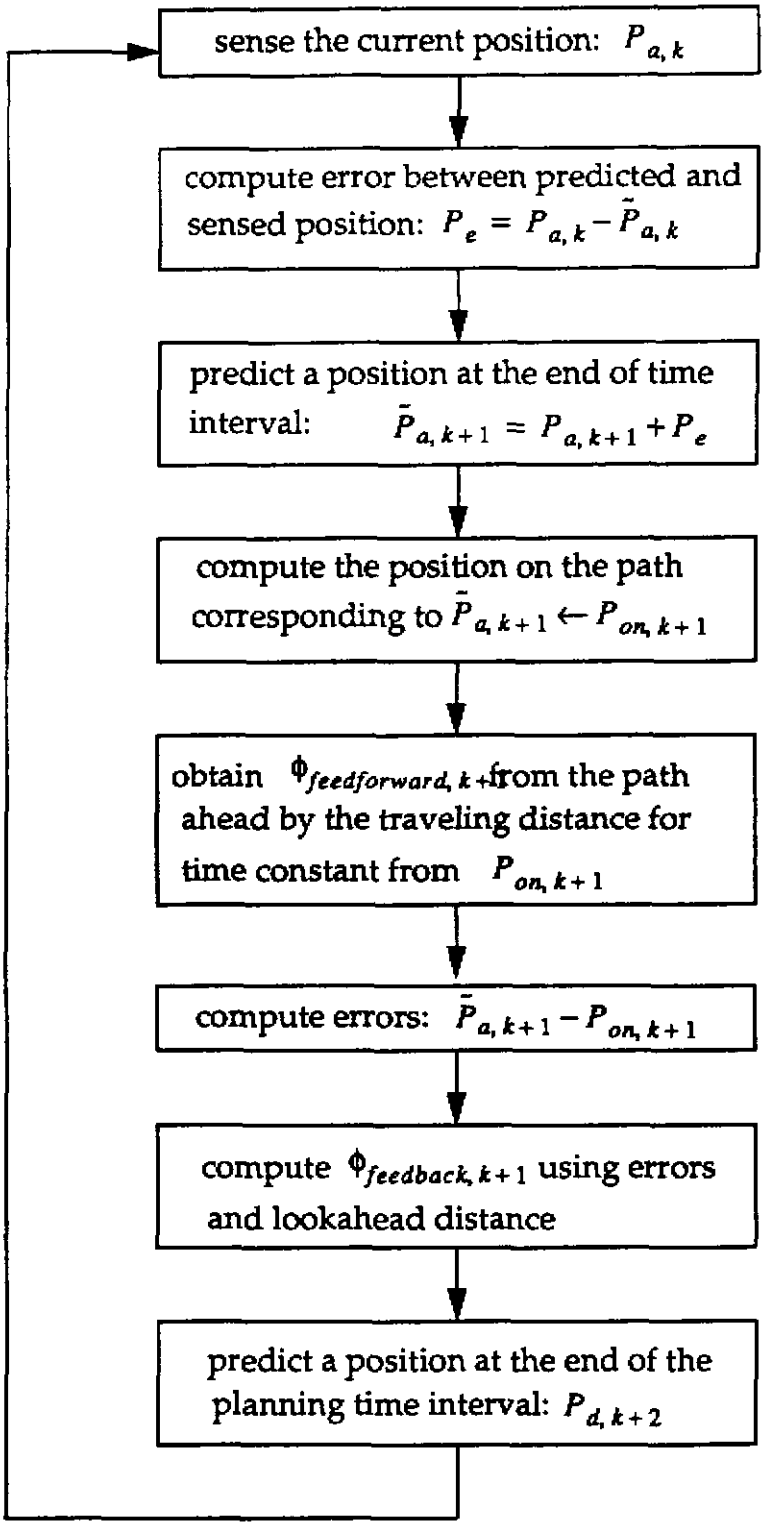

Figure 17: A flowchart of steering planning

the vehicle is sensed. Thus, in this method, the vehicle position at the end of the planning time interval (16 mS) is predicted rather than the position at the end of the sensing time

interval (250 mS). Figure 18 illustrates the timing used in this approach.

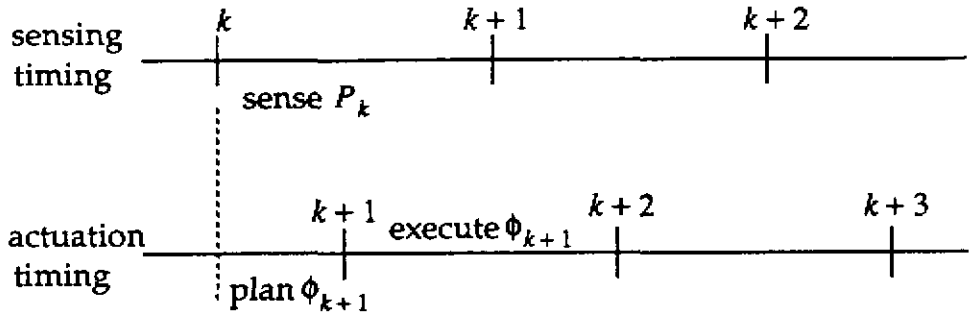

Figure 18: Separate sensing and actuation timing

2.2.6. Speed Planning

As mentioned in Section 2.3.2, one of the benefits of choosing the guide point to be the center of the rear axle is that steering and propulsion can be controlled separately. Furthermore, only steering control decides how well an intended path is tracked in geometric "path" tracking problem which concerns only deviation errors lateral (not longitudinal) to an intended path. Thus, speed (propulsion) is controlled on the basis of the following constraints:

- maximum speed limit in a particular area
- distance to possible obstacles and the destination
- path curvature, i.e. for constant lateral acceleration, curvature of the section of the path immediately in front of the vehicle can be used to constrain the vehicle speed as the following:

$$a_{lateral} = \frac{v^2}{r} = cv^2 \tag{28}$$

Speed can be chosen to be the minimum of the above constraints and this speed is sent as input to the propulsion servo control.

2.3. Results

In order to show performance improvements by the feedback and feedforward schemes developed in Section 2.3, the following four simulations are considered:

- The vehicle is made to follow a path in open-loop fashion. In this case only the steering angle is regulated, while vehicle position errors are not compensated (Figure 19).
- Vehicle position errors are compensated using only the feedback scheme discussed in Section 2.3.4 (Figure 20).
- The only compensation made is a feedforward one, using the control scheme

discussed in Section 2.3.3 (Figure 21).

- Both feedback and feedforward schemes used in Figure 20 and Figure 21 are applied to compensate for the vehicle errors (Figure 22).

In the open-loop case, the resultant path is often parallel to the desired path. This is because, after the desired curvature is obtained, no further correction is made unless the input is changed. In the case of feedback only, the vehicle oscillates around the nominal path because the feedback gain makes the vehicle sensitive to position errors. For lower value of feedback gain (longer look-ahead distance), much less overshoot is noticed, but the feedback response is very sluggish. In the feedforward only case, the algorithm was able to compensate for the vehicle in a predictive fashion. However, such errors accumulate without feedback in the general case. In the last case, it can be seen that the combined feedback and feedforward schemes complement each other and the resultant path is qualitatively superior to those obtained by application of the other methods.

Experiments have shown results similar to those in the simulations. Figures xx show graphical results from real time navigation of the Navlab with variations of feedforward compensating time. The desired path consists of a 20 *m* straight line, a sudden jump by 5 *m* in lateral direction, and an 80 *m* straight line from the initial offset to see the step response. Only with feedback compensation alone does the actual vehicle path shows serious oscillation, as in Figure 23, which is similar to the simulation result of Figure 20. When the feedforward compensation time was 0.4 seconds (Figure 25) or 0.6 seconds (Figure 26), the experimental results showed the best tracking performance. These figures indicate that the Navlab has inherent latency of 0.5 seconds. Note that the look-ahead distance in the feedback compensator was fixed at 15 *m* to concentrate on the feedforward compensator evaluation. Evidently, the feedforward compensation has a dominant effect on the tracking performance, as the simulation and experimental results (Figure 22 and Figure 25) show dramatic improvement from those without feedforward compensation.

Figure 27 and Figure 28 show tracking performance on paths driven by Navlab at the Pittsburgh zoo parking lot. The length of paths are more than 500 *m* and maximum speeds are more than 20 km per hour. Note that there are some deviations from an intended path near the ends of the courses. These are due to a systemic periodic drift in the inertial positioning system- in reality the path of the vehicle was quite smooth and within the +/- 1m tolerance.

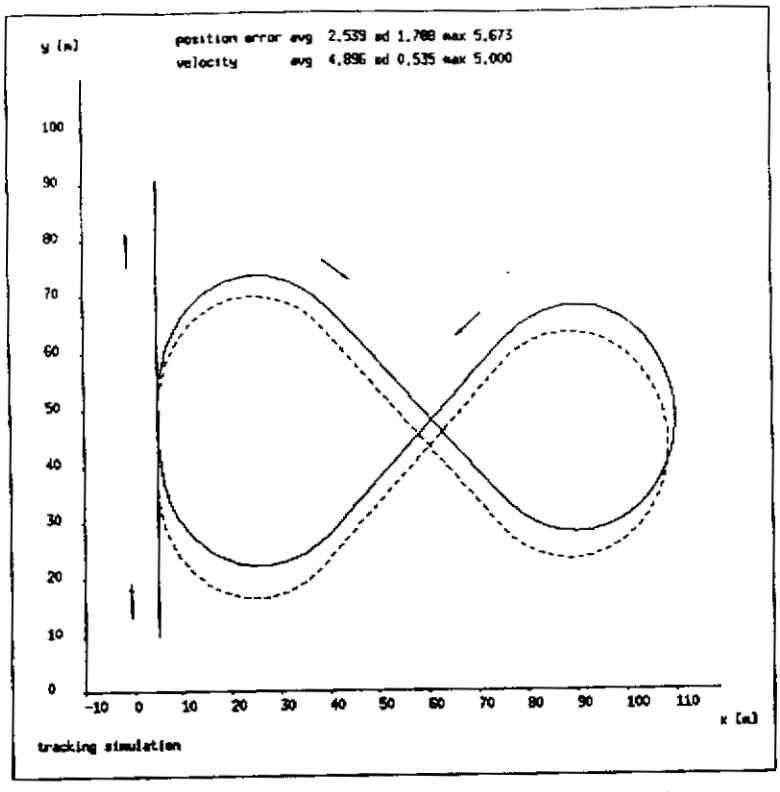

Figure 19: Tracking simulation without compensation of position errors
time constant = 1 sec, desired speed = 5 m/s, L = 15 m, planning time = 0.2 sec

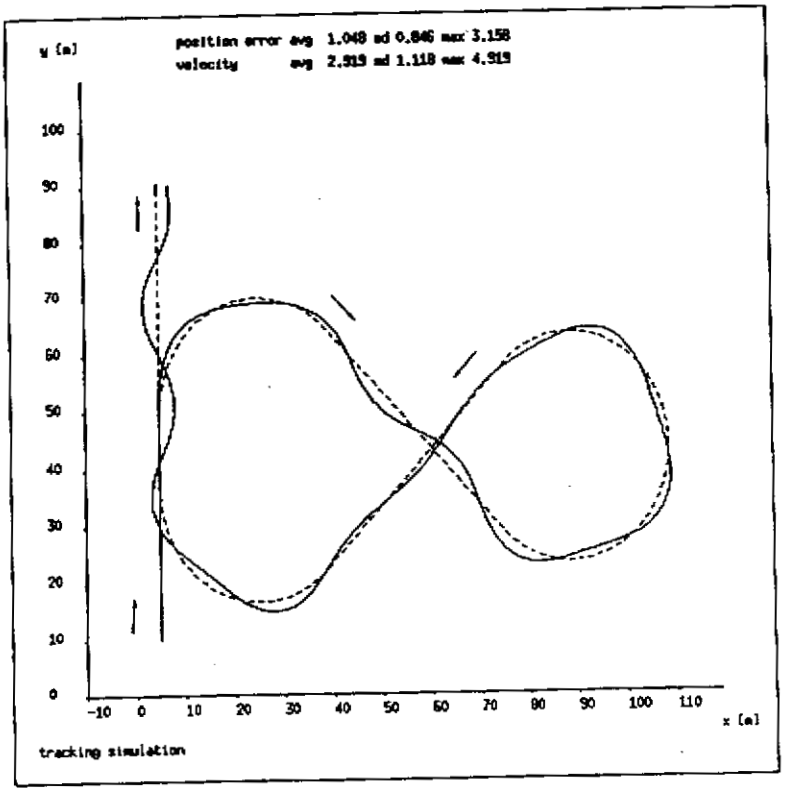

Figure 20: Tracking simulation with only feedback compensation
time-constant = 1 sec, desired speed = 5 m/s, L = 15 m, planning time = 0.2 sec

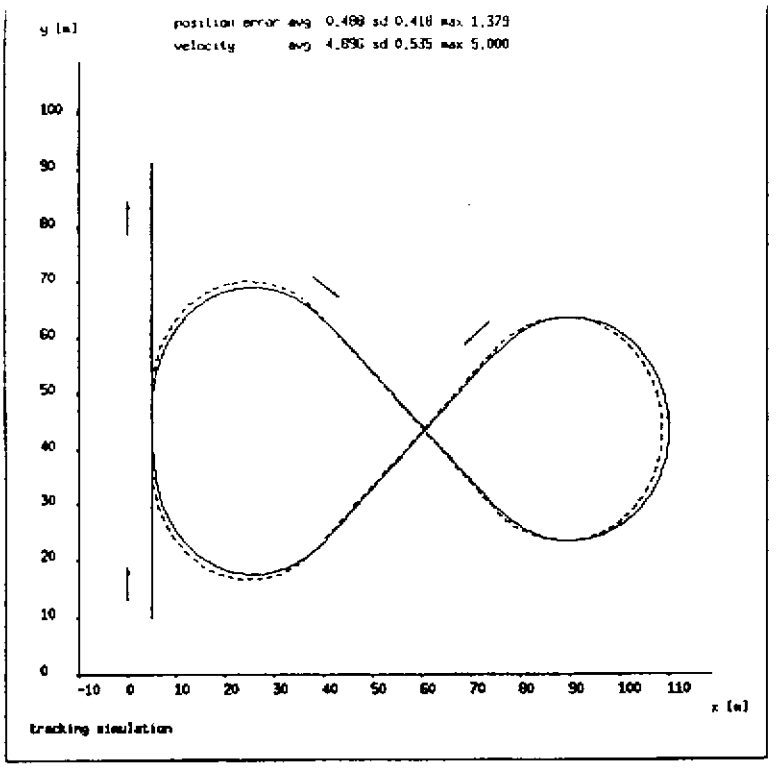

Figure 21: Tracking simulation with only feedforward compensation
time-constant = 1 sec, desired speed = 5 m/s, L = 15 m, planning time = 0.2 sec

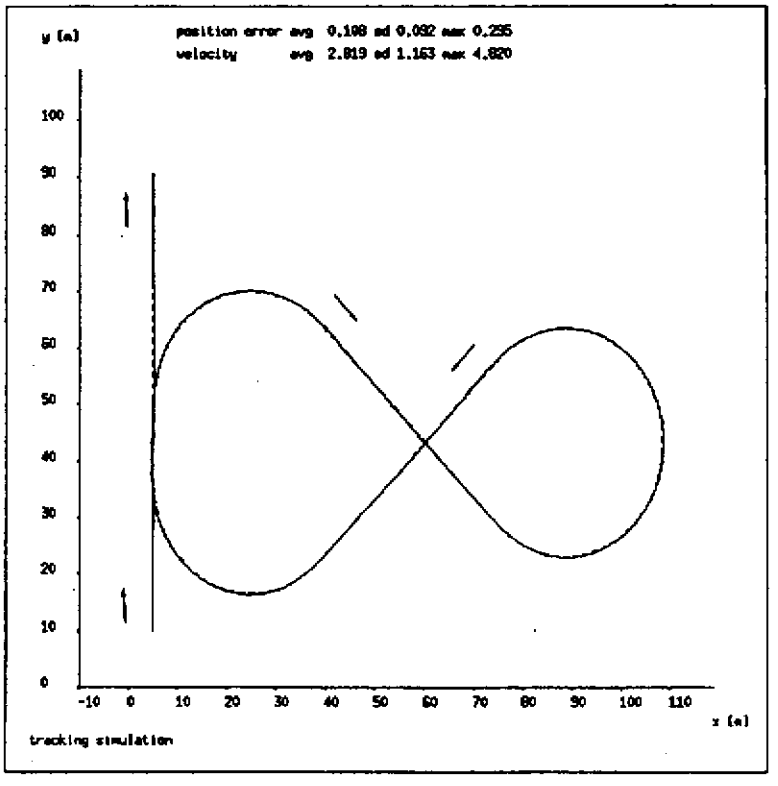

Figure 22: Tracking simulation with feedback and feedforward compensation
time-constant = 1 sec, desired speed = 5 m/s, L = 15 m, planning time = 0.2 sec

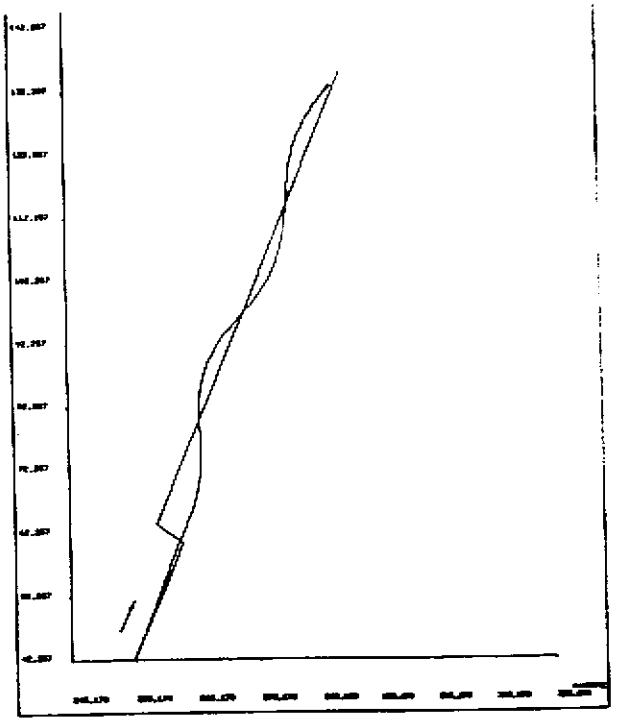

Figure 23: Real-time tracking with only feedback compensation
desired speed = 5 *m/S*, L = 20 *m*, planning time = 0.2 second,

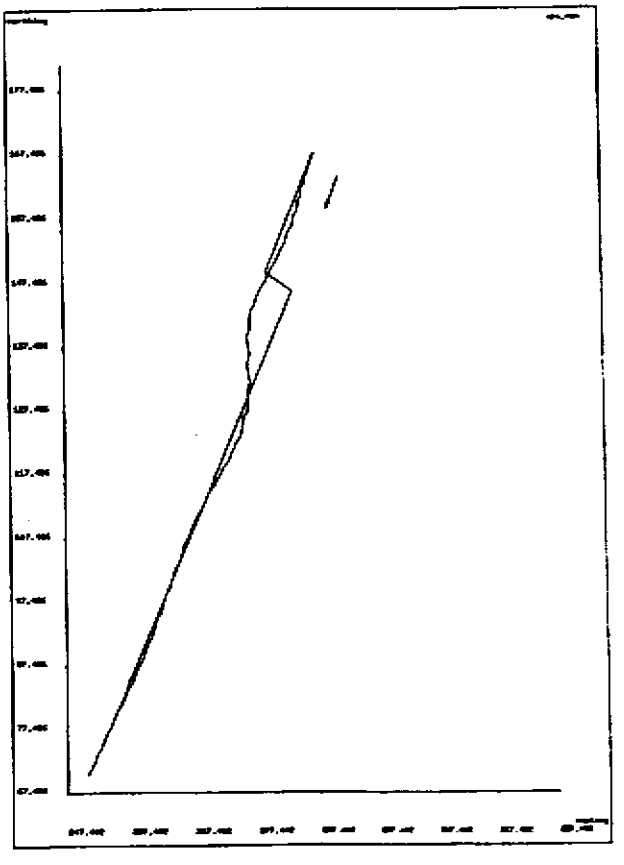

Figure 24: Real-time tracking with feedback and feedforward compensation
desired speed = 5 *m/S*, L = 20 *m*, planning time = 0.2 second
Feedforward Compensation Time = 0.2 second

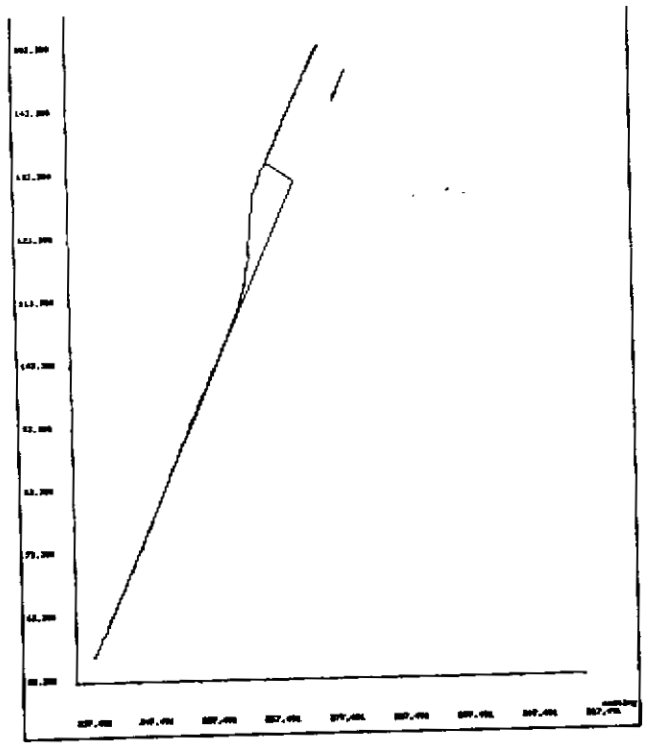

Figure 25: Real-time tracking with feedback and feedforward compensation
desired speed = 5 *m/S*, L = 20 *m*, planning time = 0.2 second,
Feedforward Compensation Time = 0.4 second

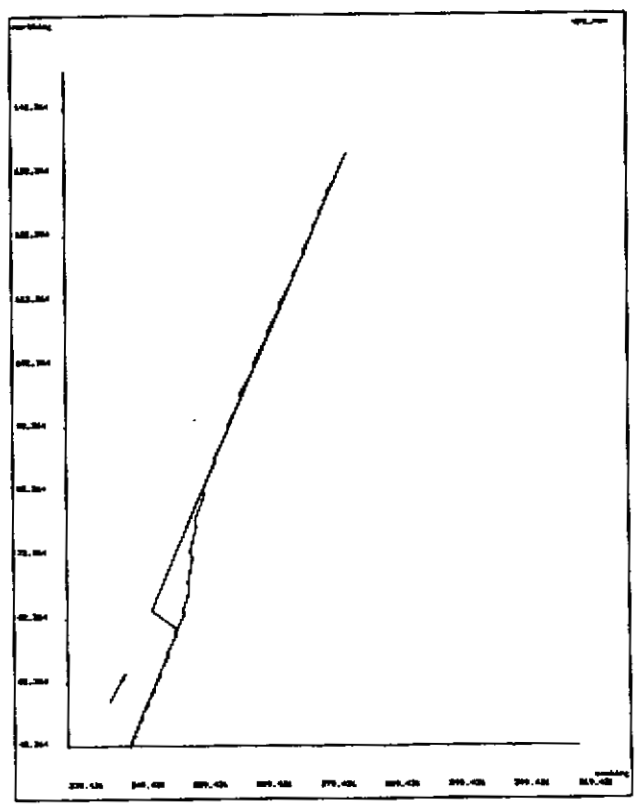

Figure 26: Real-time tracking with feedback and feedforward compensation
Feedforward Compensation Time = 0.6 second

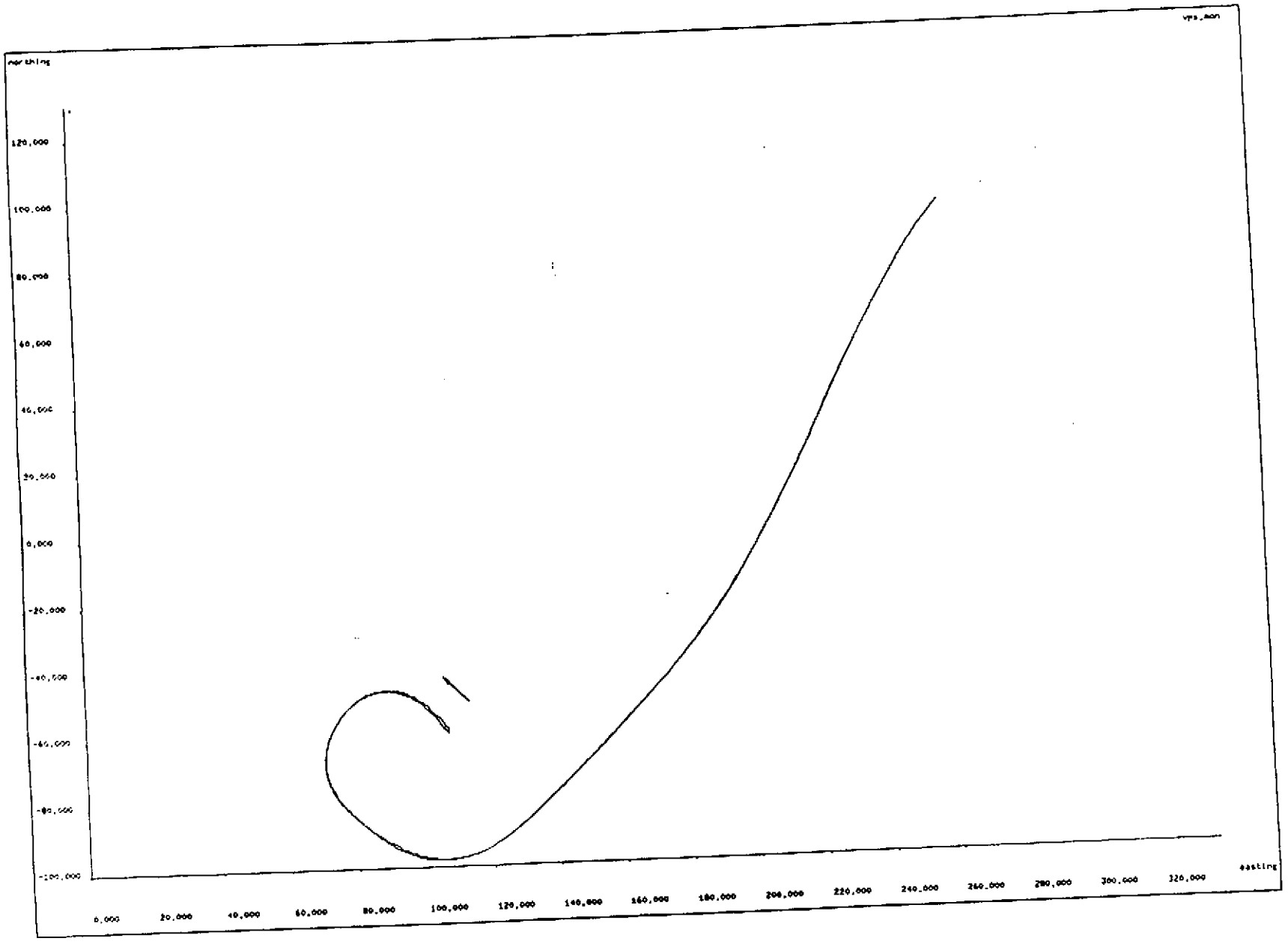

Figure 27: Real-time tracking at high speeds, planning time = 0.2 sec, L= 15m,

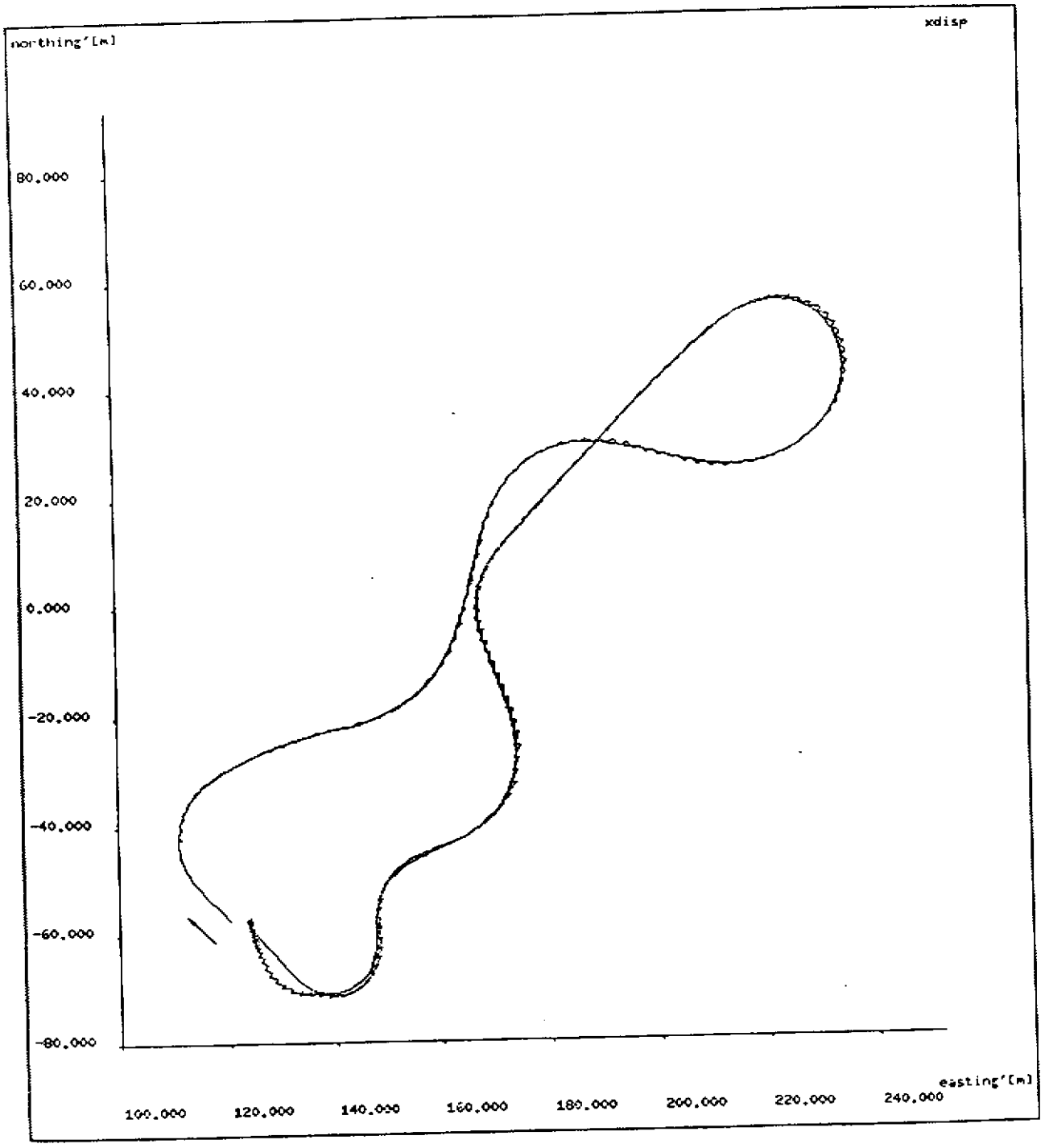

Figure 28: Real-time tracking along an instantly recorded arbitrary path, desired speed = 5 *m/S*, planning time = 0.2 second, ff compensation = 0.4 second

2.4. Path Tracking- Conclusions

This section has presented a mode of high-speed autonomous navigation for full-sized outdoor vehicles. This mode differs from precedent mobile robot path tracking in that it follows explicit outdoor paths using accurate, explicit robot vehicle positions at a high rate.

Together, the following three strategies distinguish the methodology for path tracking developed in this work: (1) geometric "path" tracking; (2) the selection of a guide point at the center of the rear axle, which exploits kinematic decoupling of steering and propulsion; and (3)multiple nested feedback control loops on actuators, vehicle geometry and path tracking error. The strategies simplify the robot vehicle tracking problem as a steering control problem. Propulsion speed can be set independently based upon criteria such as proximity to obstacles. Furthermore, the nested feedback loop strategy decomposes the problem into a steering actuator servo control problem and an on-line steering planning problem.

A partitioned scheme for steering planning is implemented in two parts: a feedforward compensator and a feedback compensator. The former guides a robot along an intended path by producing, with the use of *a priori* knowledge of the future path, an anticipatory control; and the latter compensates for errors by finding a curve which will converge with the desired path while insuring a smooth reduction of all errors; *the quintic polynomial* scheme. This method does not require a complete modeling of the full-sized vehicle system which, in any case, defies complete and precise description. Rather, it succeeds with a first order model by compensating for the total system latency, which appears to be the dominant characteristic in the behavior of full sized vehicle. This circumvents the problem of complex physics in modeling and control and yet provides high performance through smooth error compensation and anticipatory control.

These developments were implemented both in simulation and on our testbed. By introducing functions in which all computer hardware dependant details were hidden, the developed codes were applicable directly to both computer platforms, either in simulation or in real time environment. Outdoor path tracking by a real-time autonomous vehicle has been demonstrated at speeds up to 35 km per hour, which is an American record for the highest speed achieved by an autonomous vehicle, but more significantly it is a first benchmark and world record for navigation by this promising navigation paradigm. Tracking performance was found in experiments and simulation to be more dramatically improved by anticipatory control, as the vehicle speed increased.

3. Obstacle Detection

We have previously proposed a paradigm for high speed autonomous navigation called *position based path tracking*. In this paradigm, a vehicle tracks an explicitly specified path by continuously making steering corrections using position feedback from accurate inertial sensors. Vehicle speed is determined by choosing the highest speed that meets all of several constraints like maximum lateral acceleration and distance to nearby obstacles. To this end, we propose several schemes for obstacle detection.

More formally, *obstacle detection* is the determination of whether a given space is clear from obstructions for safe travel by an autonomous vehicle. Specifically there are three goals: to detect obstacles in time when they exist, to identify the correct obstacles, and, to not be fooled by objects that are not in the path of the vehicle.

Previous work in the area has been mostly confined to robots moving at slow speeds because the task of perception is usually to determine the cues necessary to navigate in a cluttered world. There are two reasons for this. Active range sensing is a commonly used modality for obstacle detection because the sensors directly supply range from the robot to the world. Active range sensors (laser scanners and sonar) are typically slow, providing in some cases, only two frames (snap shots of the world) per second. Secondly, irrespective of whether active sensing or passive sensing (reflectance images) is used, an enormous amount of data must be processed in real-time to discern obstacles in the world. Hence, successes in this area have been limited to speeds between 2-3m/s even though very sophisticated laser ranging and systolic array processors have been used for sensing and computation[9, 4, 15]. More recently, some researchers have demonstrated obstacle detection using passive vision along with special purpose hardware at speeds up to 14m/s on straight stretches of flat highway[21]. Passive vision, however, is limited in scope because it is prone to poor illumination and low contrast conditions.

Our approach has been two fold. Firstly, we use accurate position estimation to determine the location of the autonomous vehicle on the path it is to follow. This enables us to narrow our search for obstacles to that part of the field of view of our sensor where we expect the future path of the vehicle to lie. Instead of finding objects in the world so that we can steer around them, we "detect" obstacles when they exist in the path of the vehicle and stop for them. Secondly, we propose the use of laser scanners that produce sparser but more frequent snap shots of the world than have been used by researchers in the past. We have used Cyclone, a laser scanner designed and fabricated in-house for our initial experiments (see also section 8.1.). Cyclone was configured to provide a single scan line (1000 points in a 180 degree field of view) of range data at up to 8 scanlines/second in our initial experiments.

We chose to begin our experimentation with the assumption that the vehicle travels in a 2-D world with the further simplification that the vehicle does not pitch. These assumptions were relaxed in order to consider a 3-D world in which it is necessary to aim a laser sensor towards the ground in front of the vehicle. We have developed two different methods. One that considers only a few scanlines from a range image, either because the laser is only able to provide sparse data or if a dense range image is available, for the sake

of efficiency. The other method explores the scenario where a dense range map is available but must be processed very rapidly. To this end, we have designed schemes similar to those proposed by Dunlay[9], but, are more efficient in the amount of computation required.

3.1. Obstacle Detection in a Flat World

We chose to begin the investigation in a two-dimensional world. Two techniques were designed with the simplification that the vehicle travels in a 2-D world. The first method is based on identifying obstacles by matching range data obtained from a range scanner on the moving vehicle, to a world model. The second method checks a bounded region around the path to be traversed for range data corresponding to objects.Figure 29 shows the range scanner mounted on the front of the vehicle.

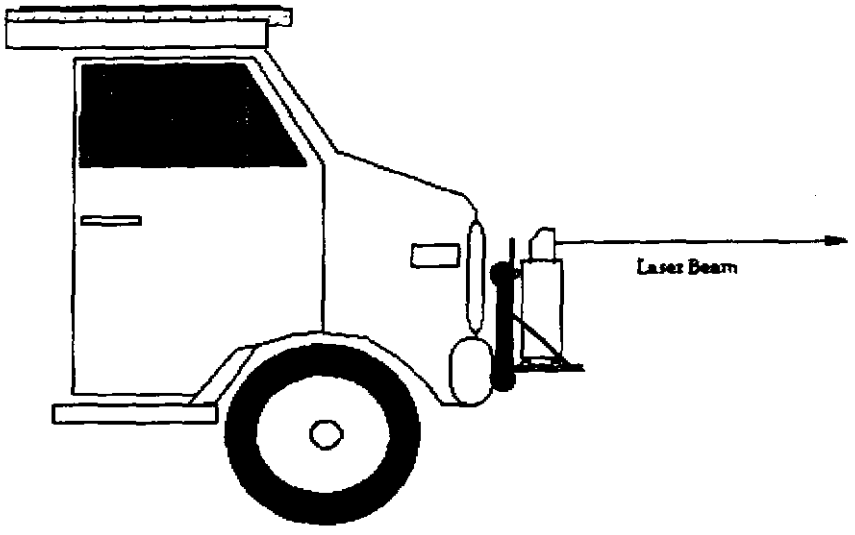

Figure 29: Scanner configuration in a flat world

3.1.1. Profile Matching

Profile matching is based on the notion of matching a hypothesized range profile using vehicle position and a world model, with a range profile inferred from range data obtained while in motion. This requires explicit modeling of the world in which the autonomous vehicle travels. Such a model could be built from range data gathered from preliminary runs or even from aerial photographs.

Figure 30 shows a vehicle travelling in open terrain. One of the range profiles shown is generated as an expectation provided that the exact location of the vehicle and a model of the environment is available. The other profile is actually generated from a scanline of range data. These two profiles are matched to determine whether or not the path ahead of the vehicle is clear. In the case shown, there is a marked difference in the expected and obtained range profiles and application of standard correlation techniques indicate that an unexpected object is within the field of view.

Although this scheme was fairly successful in simulation, we noted the following:

- The real world is very hard to model. Not only is the world dynamic, but exact calibration is a difficult task. Building and maintaining a consistent world model is often a weak link in an actual implementation.
- Profile matching can only detect if something has changed in the environment but cannot distinguish between unexpected objects that are on the road ahead and those that are clear of the vehicle path. A further level of processing is necessary to determine if the unexpected obstacle might interfere with the vehicle.
- Errors in position estimation result in errors in the matching process, even with perfect range data.

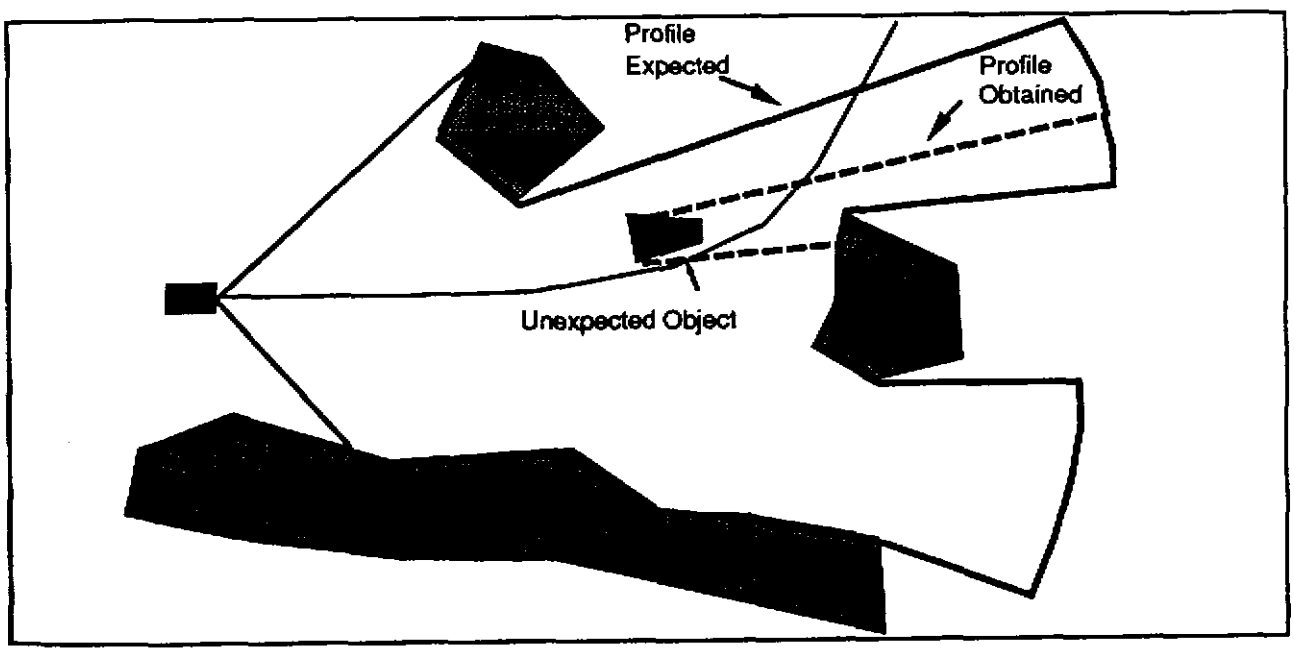

Figure 30: Profile matching

3.1.2. Clearance Checking

The above observations led us to focus our attention on the path ahead rather than examining the entire field of view. More precisely, we are interested in the space which the vehicle could possibly sweep out in its future travel. A simple method is to search a section of the path immediately ahead of the vehicle for obstacles. For a vehicle to travel unimpeded, no objects should lie in this area. This method accrues the following advantages:

- There is no need to model the world. There is an implicit model: the road ahead should be clear of obstacles.
- Any objects detected in the path of the vehicle are immediately relevant.
- Errors in position estimation result in the vehicle deviating from the reference

path by the amount of the error. However, since the tracking and obstacle detection schemes use the same position feedback, the same deviated path is searched for obstacles.

The space that the vehicle could possibly sweep out in the near future (collision zone) is identified periodically and range data is checked to see if any of the corresponding reflecting points lie within the zone. Figure 31 shows a scenario in which the clearance checking scheme isolates range measurements due to an object within the collision zone.

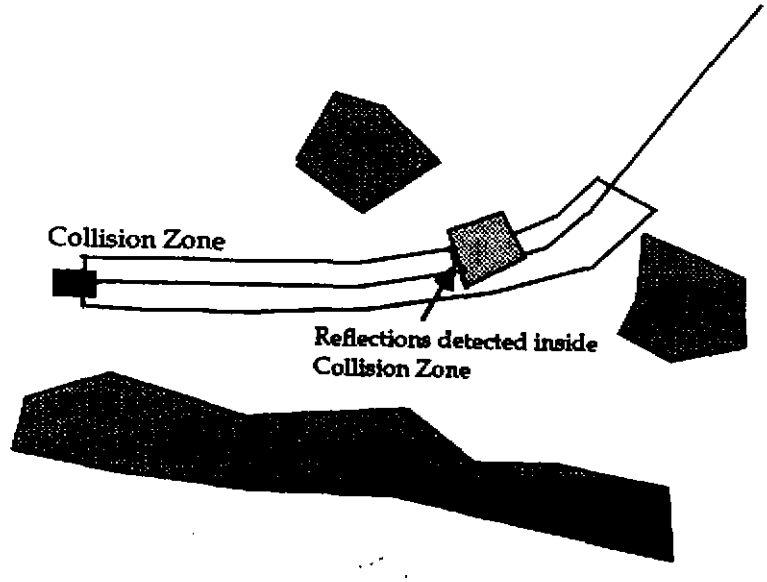

Figure 31: Obstacle detected inside the collision zone

Clearance Checking is given an explicit specification of the path on which the vehicle is to travel in the form of a sequential list of posture- a specification of position, orientation and curvature that the vehicle is to attain. In addition, vehicle position is available by querying an inertial navigation system. The procedure is initiated whenever a scanline of range data becomes available. Then, the following steps are executed:

1) The current vehicle position is obtained and the current scanner position S_p is computed. P_k, the posture on the reference path closest to S_p is deter-

mined as the posture with the smallest lateral distance to S_p (Figure 32).

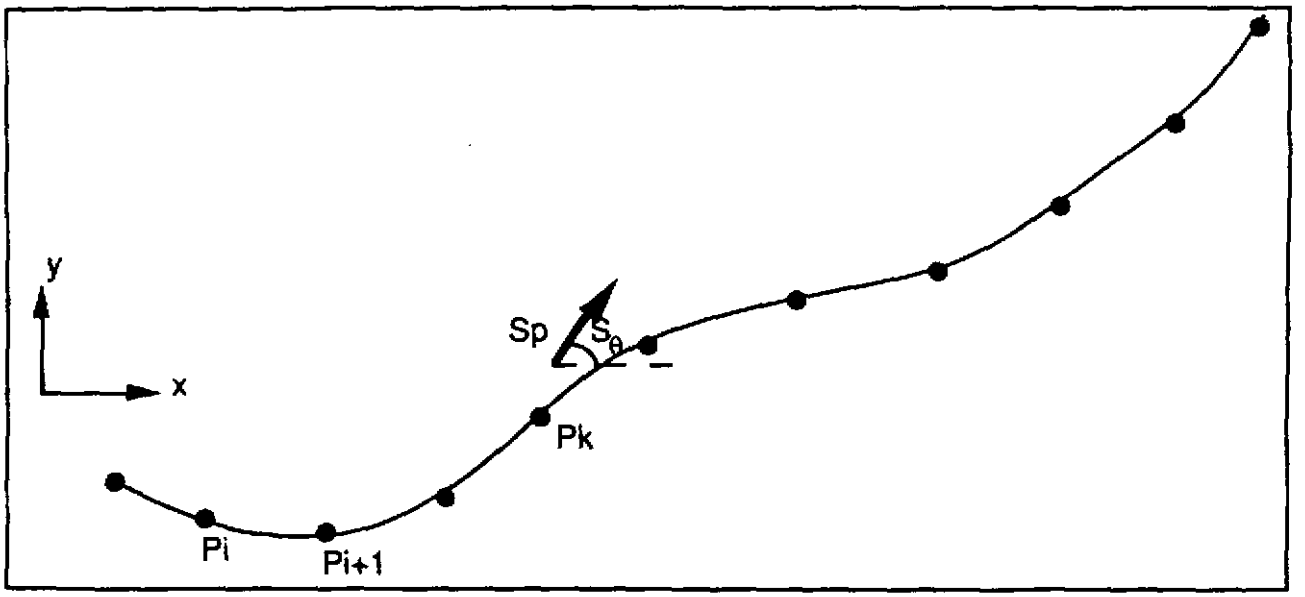

Figure 32: The path in global coordinates

2) n future postures of the vehicle are transformed into vehicle coordinates using a transformation from global to local coordinates. Now, the list of postures local to S_p is given by p_j (p_{jx}, p_{jy}, $p_{j\theta}$) in the following manner:

$$p_{jx} = (P_{i_x} - S_x)*cos(S_\theta) + (P_{i_y} - S_y)*sin(S_\theta) \quad (29)$$

$$p_{jy} = (P_{i_y} - S_y)*cos(S_\theta) + (S_x - P_{i_x})*sin(S_\theta) \quad (30)$$

$$p_{j\theta} = S_\theta - P_{i\theta}\ (i: k+1 \ldots n;\ j = i-k) \quad (31)$$

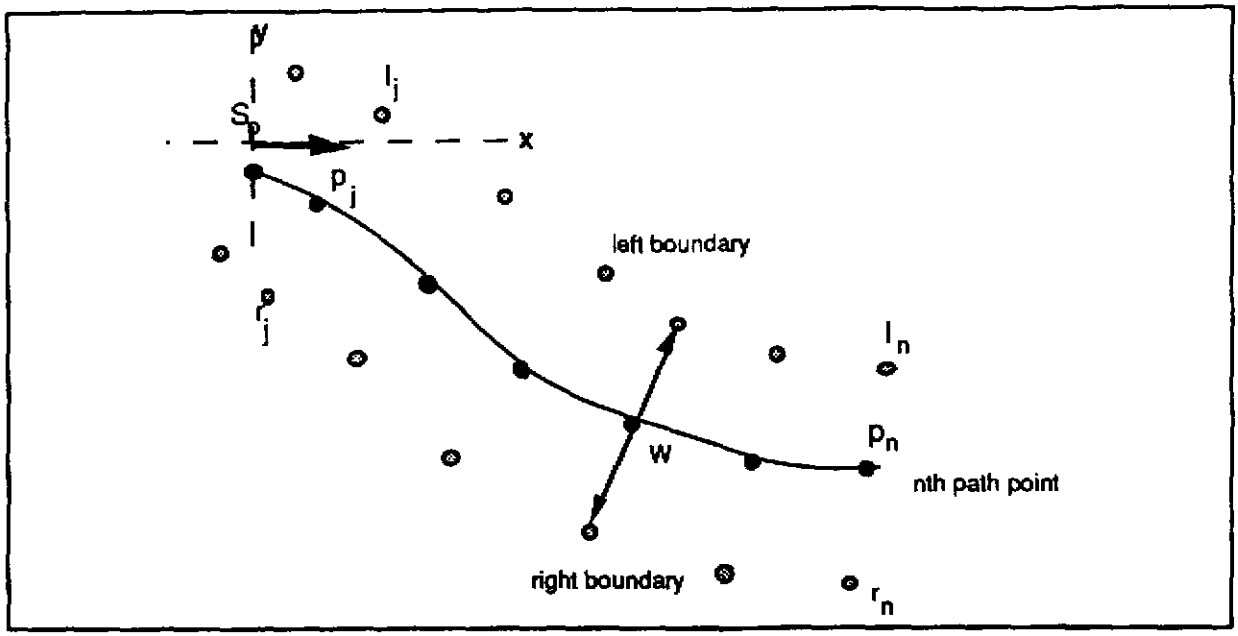

Figure 33: The path and edges in local coordinates

3) The locations of the left and right edges of the collision zone are computed in local coordinates using W, the total width of the collision zone, as shown in Figure 33. $W = W_v + 2(S_\varepsilon)$ where W_v is the largest dimension of the vehicle and S_ε is the maximum error in position sensing. The coordinates of the left and the right edges of the road (in local coordinates) corresponding to p_j are given by r_j and l_j respectively:

$$r_{jx} = p_{jx} + \sin(p_{j\theta}) * W/2 \tag{32}$$

$$r_{jy} = p_{jy} - \cos(p_{j\theta}) * W/2 \tag{33}$$

$$l_{jx} = p_{jx} - \sin(p_{j\theta}) * W/2 \tag{34}$$

$$l_{jy} = p_{jy} + \cos(p_{j\theta}) * W/2 \tag{35}$$

4)The length of the collision zone, C_l is determined by taking the minimum of a preset length and the distance along the x axis to a point ahead where the path is oriented more than 90 degrees relative to S_p.

5) A distance C_l ahead of the vehicle is tessellated into m intervals (0.5 m long) and the collision zone boundaries are transformed into a hash table indexed by fractions of this distance. For each of the m intervals, a left and right edge point of the collision zone is found through interpolation (l'_h, r'_h) as in Figure 34. The reason for the 90 degree restriction mentioned above is that if the path turns more than that amount, it is not possible to find a unique bracket (l'_h, r'_h).

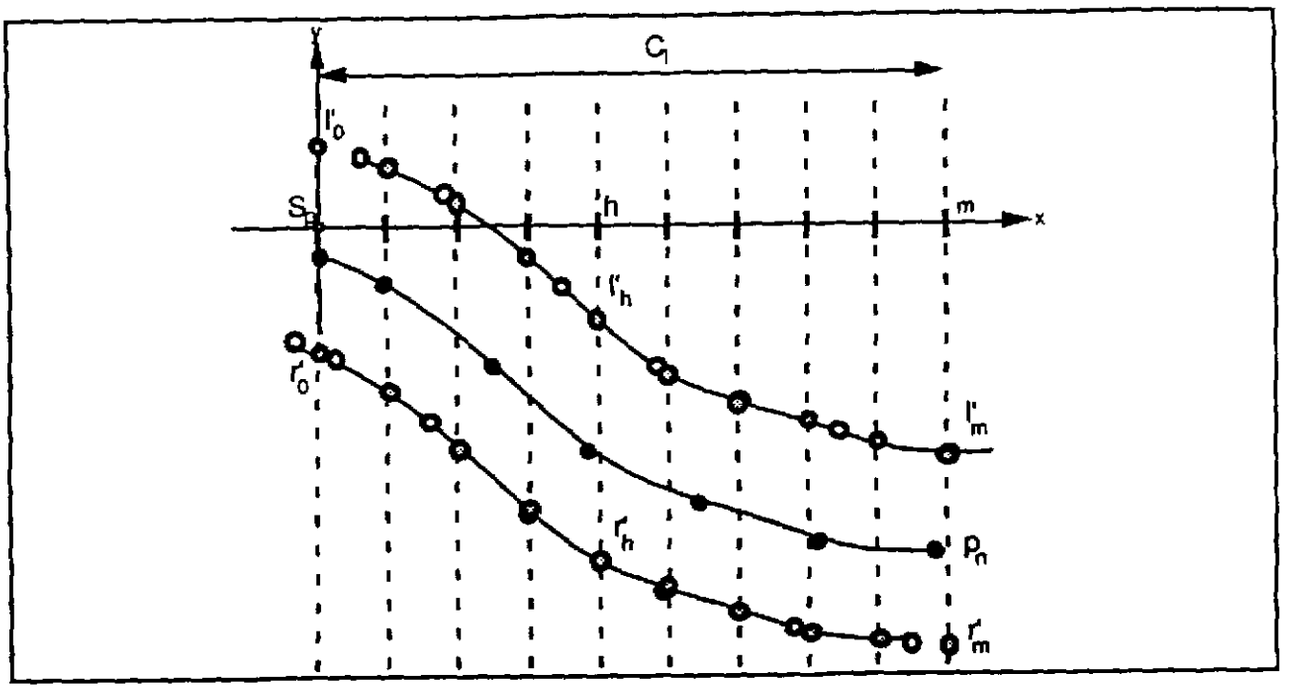

Figure 34: The collision zone stored as a hash table

6) Every range point in the scanline of range data is converted into cartesian coordinates (x_r, y_r) with its reference frame located at S_p. x_r is used to obtain the index into the hash table, k_r.

7) A range point lies inside the collision zone if $l'k_r > y_r > r'k_r$. An obstacle is indicated if more than a preset number of range points are found within the collision zone.

Experiments with a target 0.5m wide mounted on a cart moving at 6m/s towards a stationary scanner, demonstrated the ability to detect and bring the cart to a halt in sufficient time. Implementation on the NavLab was partly limited by the ability to decelerate the vehicle; we were able to stop for similar targets at speeds of up to 3m/s. However, some limitations were noted:

- Since the scan is horizontal to the ground plane, obstacles that are lower than the height of the beam are not detected.
- Pitch motion of the vehicle as well as graded surfaces pose problems because reflections from the ground are interpreted as obstacles, or conversely the beam completely misses legitimate obstacles.
- It is possible that no points are found in the collision zone due to occlusions caused by objects that are outside the collision zone. For example, if the path bends around a corner, an object on the inside of the corner, but off of the path could occlude objects that lie on the path. To absolutely guarantee a

collision free path a conservative estimate has to be made as to the length of the collision zone every time the path turns.

3.2. 3-D Obstacle Detection Schemes

As indicated above, range scans parallel to the ground plane are not sufficient to detect obstacles for a vehicle travelling at high speeds. Rather, we found it necessary to angle the scanner down toward the ground in front of the vehicle. This perspective allows for consideration of a 3D world as well as range scanners that produce multiple lines of range data.

Conceptually, the extension to 3-D is a fusion of the schemes discussed above. Here we have attempted to model the road surface on which the vehicle is to travel. In addition, the collision zone idea is used to limit treatment of only the relevant part of the world - the area that is swept out by the vehicle as it travels. The selected data is compared to a road model. Obstacles are indicated by deviations of range data from a road model.

Two variations of the 3-D schemes were considered:

- Selected Scan Method - a few scanlines from a complete range image comprised of multiple adjacent scanlines are selected, and an elaborate fit of the road profile (a cross section of the road transverse to the direction of travel) is developed. Roads are assumed to be graded or designed roads, generally smooth and with crowns and/or banks.
- Full Frame Analysis - the entire range image is used and a simpler fit is performed on all the range data corresponding to the road width being verified. Roads are assumed to be graded but neither banked nor crowned. This approach is only tractable when multiple lines of range data and sufficient real-time computing resources are available.

The following assumptions were made to limit possible scenarios:

- "obstacles" have a visible face of 0.5 meter X 0.5 meter,
- road grades are less than 11%; rate of road grade at less than 1%/meter,
- bank and crown slopes are of less than 5%,
- maximum vehicle speed is 6 meter/second.

The schemes had to take the following into consideration:

- Non-Ideal Vehicle Motion: In reality, the NavLab can pitch +/- 5 degrees. However, this motion is of a low frequency (0.2 to 1 hz) and can be compensated for by using a low pass filter on the pitch angle to get an estimate of instantaneous pitch. This estimate can be use to select an appropriate part of the range image if multiple scan lines are available or by using a higher scan rate if only a single scanline is available.
- Undulating Terrain: One simplification is to consider the world as being locally flat. However, the extent to which this is successful is a function of how fast the robot moves, simply because the stopping distance is increased at higher speeds and the distance the robot has to scan ahead of the vehicle is

also increased.

- Field of View: The field of view limits the stopping distance because as the path ahead of the vehicle turns, one of the edges of the road falls outside the field of view of the scanner. In this case, the furthest distance at which both road edges can be seen is used to limit the safe-distance up to which the vehicle can travel without interference. In our experience a field of view of 90 degrees is necessary for the speeds that were targeted.

3.2.1. Scanner Model

Figure 35 shows a side view and a top view of the scanner model used.

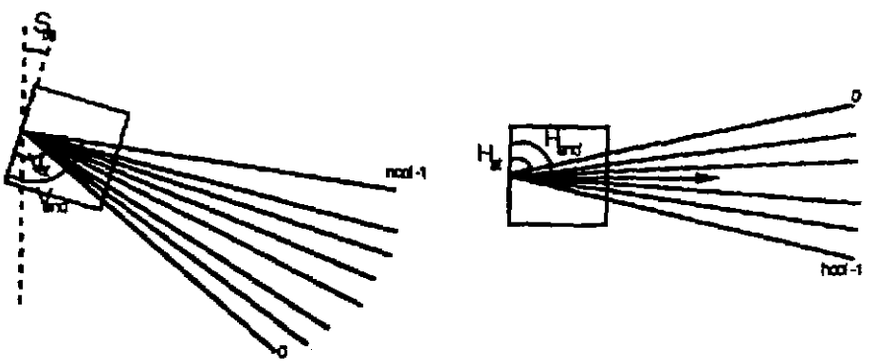

Figure 35: A generalized scanner model

A point in the world (x, y, z), with respect to the scanner coordinate frame can be projected into an image plane to obtain the azimuth and vertical angles (θ,ϕ) in the following manner:

$$\theta = \text{atan2}(y,x) \tag{36}$$

$$\phi = -\text{atan}(z - S_z/x) + \pi/2 \tag{37}$$

where S_Z is the height of the scanner from the ground plane.

Given a discrete image plane, as in one that has a finite field of view and resolution, row and column numbers for the point corresponding to the angles (θ,ϕ) are given by:

$$row = [(\phi - V_{st} + S_{tilt})/(V_{end} - V_{st})] * (nrows - 1) \tag{38}$$

$$col = [(\theta - H_{st})/(H_{end} - H_{st})] * (ncols - 1) \tag{39}$$

where S_{tilt} is the tilt of the scanner, H_{st}, H_{end} are the starting and the ending angles in the azimuth axis, V_{st}, , V_{end} are the starting and the ending angles in the vertical axis, and, *nrows* and *ncols* denote the number of pixels in the horizontal axis and the vertical axis, respectively.

3.2.2. Using Selected Scanlines

This method selects to process only a few but relevant lines, based on vehicle pitch and vehicle speed. The motivation for this method is two fold. Firstly, searching an area in front of the vehicle for obstacles usually reduces to finding obstacles in a small subset of the vertical field of view. Secondly, such a method allows for a wide range of scanner configurations: from a single line scanner to an arbitrary multiple line scanner.

A second order polynomial is fit to the height profile of each useful scan line. That is, an estimate of average height, bank and crown is made for each processed scanline. The algorithm is recursive so that a single estimate is saved for bank and crown to represent all past data. When a new height profile is considered, it is matched against the road model that has been built up from previous scanlines of range data. The amount of weight to be placed on old data as opposed to new data can be adjusted with a trade-off between noise tolerance and sensitivity to terrain changes. That is, in the case that old data is weighted more, the algorithm is more tolerant to noise, but also becomes less "adaptive" and sharp changes in the terrain can be erroneously interpreted as obstacles.

3.2.2.1. Method

First, as in Clearance Checking, the road edges are determined. However, the hash table constructed before is not useful; we will process the data differently. When a new frame of range data is available the following steps are taken. Steps 1, 2, and 3 are the same as in the case of Clearance Checking.

4) Based on the current speed and an assumed deceleration of the vehicle, a stopping distance D_S is computed. If the first scanline that can be used (the one with the steepest elevation angle), falls farther than the stopping distance, an exception is signaled and the vehicle is brought to a halt.

5) The distance that the vehicle will travel in the time that it takes to complete one full scan is computed. This distance is divided by the number of scanlines n to be examined during this interval, yielding n equally spaced road locations. The magnitude of n is based on the maximum speed of the vehicle, the size of the smallest obstacle to be detected and the computing resources available. The row numbers that correspond to these road locations are then identified as in (38) for extraction from the full scan. This process is shown in Figure 36.

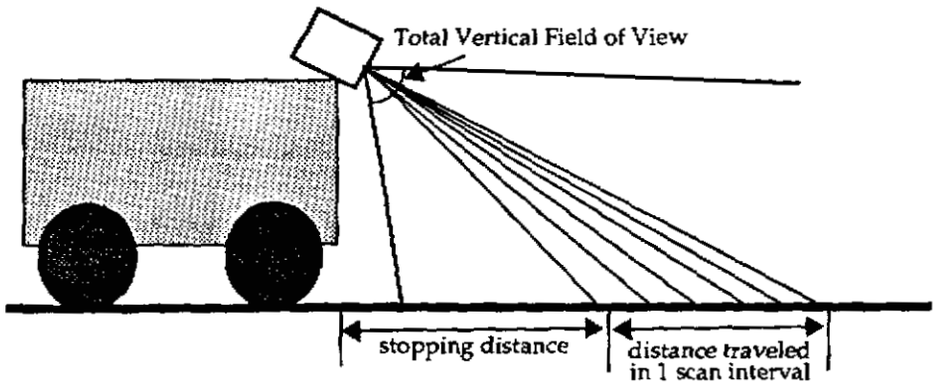

Figure 36: Scanline distribution

6) A spline interpolation is performed to define the left and right road edges (in global coordinates) at any arbitrary distance in front of the vehicle. For each of the n lines chosen in step 5, the points of intersection of the scanline with the left and right edges of the road are determined in global coordinates (r_x, r_y) & (l_x, l_y).

7) These edge points are projected onto the image plane, giving the pixel column numbers corresponding to the left and right edges of the road (l, r) as given by (39).

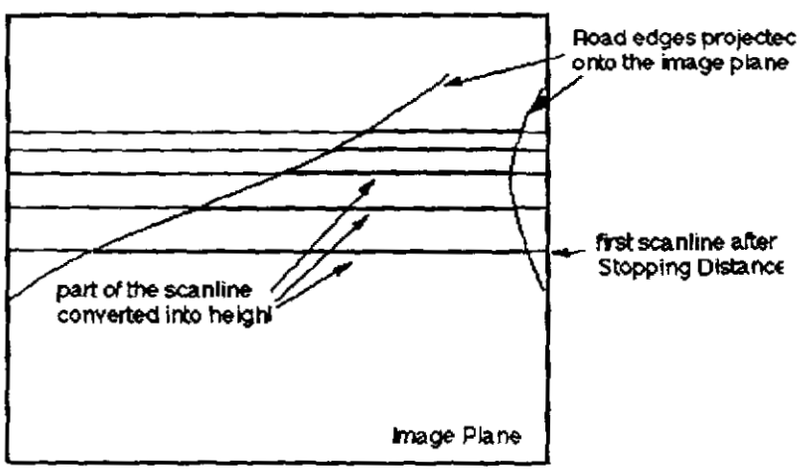

Figure 37: Selected scan lines in the image plane with delimited sections corresponding to the road surface

8) A delimited scanline is created by selecting only those pixels which lie between l and r. Figure 37 shows the view of the scanlines from Figure 36 in the image plane.

9) The stopping distance is set as the maximum distance at which both edges of the road are visible within the bounds of the image plane.

10) Each delimited scanline of range values, r_i is converted into a height profile Z_i using a transform from cylindrical to cartesian coordinates:

$$Z_{i-l} = S_Z + r_{i-l} * \cos(\theta_{i-l}) * \cos(\phi_{i-l}) \quad (40)$$

11) This profile is processed to identify unknown objects in two different ways. First, the height profile Z_i is convolved with a 7 point function U (first derivative of a gaussian) to obtain Y_i in the following manner:

$$Y_i = \Sigma (Z_{k-j} * U_j) \quad (j = -3...3) \quad (41)$$

$$U_j = [-0.5, -1, -2, 0, 2, 1, 0.5] \quad (42)$$

The resulting convolution Y_k, is very sensitive to rapid changes in the height profile and gives a very high value in the neighborhood of "edges" in the profile. Obstacles are found by looking for values of the convolution greater than a preset threshold (determined by computing the value of the convolution for the maximum expected noise).

The second method filters the height profile to obtain a recursive weighted second order fit to estimate the shape of the profile. Average height is computed from each profile, but bank and crown are computed based on previous data. Sensitivity to new data can be tuned by reducing the weights corresponding to old data. The weighted road profile is checked against the actual road profile and deviations that exceed a preset threshold are counted as belonging to an obstacle.

12) If an obstacle is found, the vehicle is brought to a halt.

13) After each frame of range data has been processed, a safe distance is sent to the vehicle controller. If this is smaller than the minimum required safe distance, then the vehicle is halted. This allows for the vehicle to start moving as soon as the obstacles are removed.

3.2.2.2. Results

Figure 38 compares edge detection and thresholding in simulation. Each frame of range data is 256 X 256 but only 10 lines of range data are selected for processing. Lines at the bottom of the figure correspond to scanlines that are close to the scanner on the ground (i.e they have the steepest elevation angles). In (a) each height profile is compared to a weighted quadratic fit that uses the recursively built up road model. The bands around each height profile indicate limits of a height threshold used. In (b), the same scanlines are checked for edges. The obstacle is detected by both methods, but the edge operator detects the obstacle about 1m before the threshold is exceeded.

Each of *edge detection* and *road model matching* has strengths and weaknesses. While edge operators are useful, edges are not very distinct in range images; for example in the case of a large object lying across the entire road region. Model matching has its problems in that the proper weighting between previous and current data must be found through

experimentation. One danger of placing too much weight on new data is that an obstacle itself influences the recursive fit making it harder to detect in future height profiles.

Three separate scenarios were tested. The first scenario was a figure-eight path on flat ground (total length 280m, the larger loop with a minimum radius of curvature of 40m, and the other loop with a minimum radius of curvature of 25m). This is shown in Figure 39. In this case the vehicle is travelling at 3m/s and the scanner provides 3 scan/s.

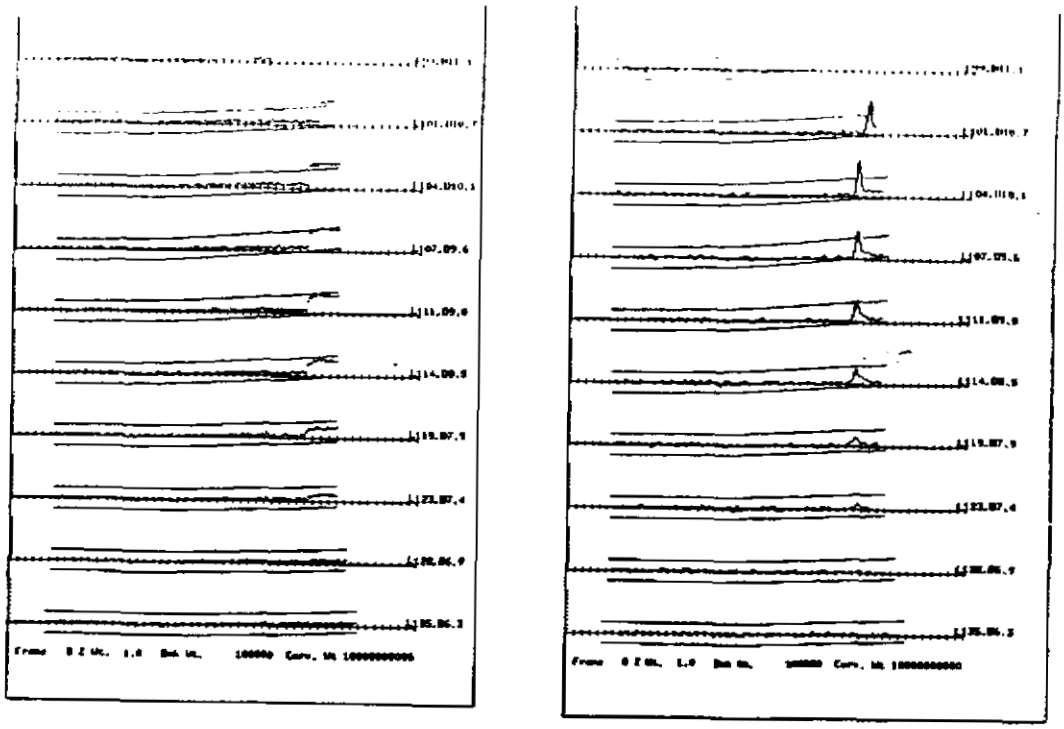

Figure 38: Comparison of (a) height threshold and (b) edge detection

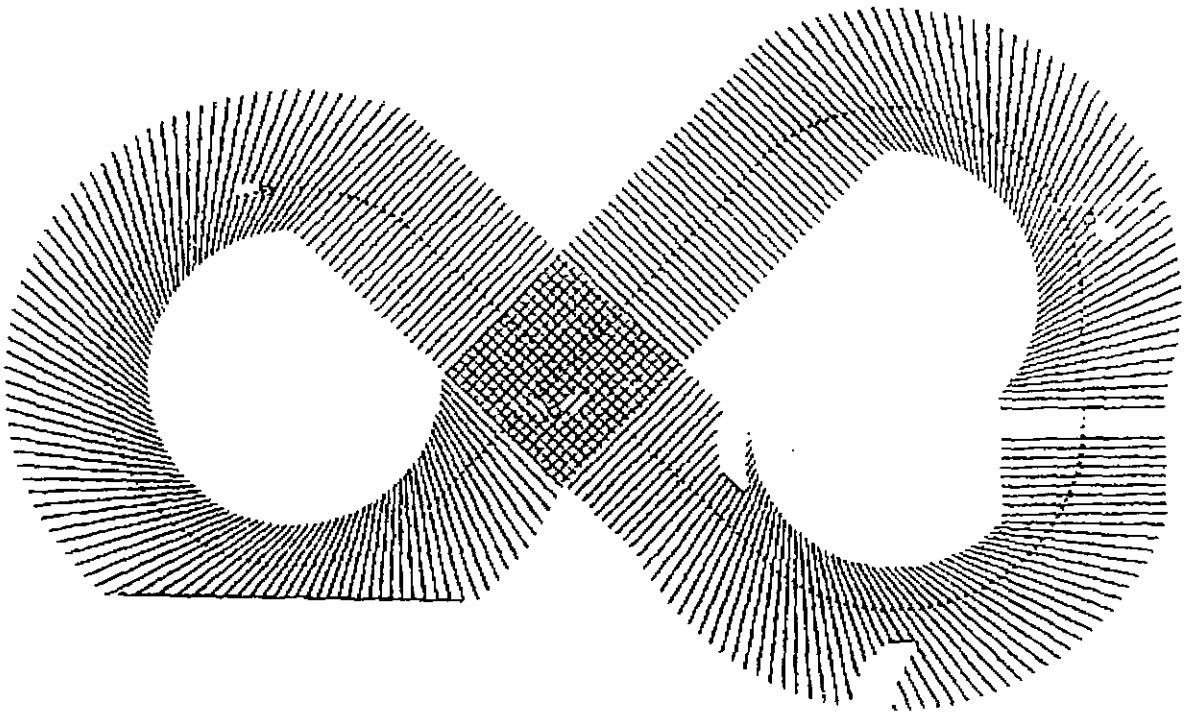

Figure 39: Scanlines on a figure 8 path

In Figure 39, the first two objects are off the road while the next 3 are on the road. In each case the correct identification was made. The other two scenarios involved straight but inclined paths with a varying transition from 0% to 10% grade. In one case, the transition occurred over a distance of 10 m, and in the second case over that of 100 m.

If only a small number of scan lines are examined from the entire frame, and especially if the extracted scan lines are chosen so that there is no overlap between the frames (as in Figure 36), then a dangerous obstacle may appear in a single scanline only. This makes the algorithm sensitive to erroneous data, that is, a single aberrant scanline could cause the vehicle to come to a complete halt. We were able to detect the obstacles, with faces of 0.5m x 0.5 m by allowing separation of scanlines on the ground to reach up to 1m. Computationally, we could process 6 scans/sec but due to NavLab's limited acceleration, we were only able to conduct successful experiments between 3 to 4 m/s.

3.2.3. Full Frame Analysis

In this approach to obstacle detection, an entire multi-line range image is processed. Obstacles are indicated by pixels in the image which lie on the road and deviate significantly from the expected road height. Groups of these pixels are gathered together into "blobs" and are treated as a unit. This process is called obstacle extraction since the final result is a blob of pixels extracted from a two dimensional array of data.

Obstacle extraction proceeds by first projecting the vehicle path into the image plane. Range data is transformed into height data and a curve is fit to the height at the center of

the road. Finally the actual road height is thresholded against the modeled height expectation and obstacles are extracted by grouping points that fall outside the threshold.

3.2.3.1. Method

In this case also, we start as in the previous two methods with steps 1, 2, 3 from Clearance Checking. The subsequent steps are discussed in detail below: In order to use all of the available data, the images must be processed at the frame rate. For this reason, most of the computations in the obstacle extraction algorithm are done in the image plane. By projecting the path onto the image, a large portion of the image can be ignored, and many needless computations avoided. Note that this is contrary to standard methods of using range data e.g., as in [9].

Assuming that the vehicle path is specified at regular intervals, the current vehicle position can be used to locate the path segment lying in front of the scanner. This path is transformed from world coordinates into image coordinates by projecting the points corresponding to the road edges into the image plane. A cubic spline is used to interpolate between the edge points. Then, for each row in the image, pixels lying between the road edges are transformed from range data to road heights; outlying pixels are discarded and not processed any further.

In the Martin Marietta system, the height of the road was determined by the height of the road at the centerline. This method fails when an obstacle lies in the center of the road. In this case the normal road height is assumed to be the obstacle height and the thresholding method suggested by Dunlay would recover "pits" (instead of obstacles) in the space between the obstacle and the road edges. If an obstacle were to span the entire road width, it would not be found at all. Therefore, we suggest a model where the grade is smoothed over a longitudinal section on the road. A third-order least-squares fit is applied to the height data at the center of each image row on the path. This has the effect of modeling the general trend of the road (up and down hills) as well as filtering out the effects of noise and small objects lying in the center of the road.

Obstacles are located by applying a height threshold on each pixel lying between the road edges. This threshold operator is referenced against the *expected* height, as predicted by the third order fit of the road centerline. In this manner, a hill is not considered an obstacle since the height expectation and the measured height should match very closely. A real obstacle does not significantly affect expected road height, due to the least squares fit and therefore is readily found by thresholding. The result of thresholding is a binary image suitable for blob extraction.

A standard blob coloring algorithm groups pixels of similar height. By grouping pixels together into blobs, the obstacles can be treated as whole units amenable to further processing.

3.2.3.2. Results

For most cases, this algorithm is an effective means of obstacle detection. Figure 40 shows a longitudinal cross section of the road. Figure 41 shows the road profile from a viewpoint local to the vehicle.The smooth curve comes from a third order polynomial fit to the

height data.

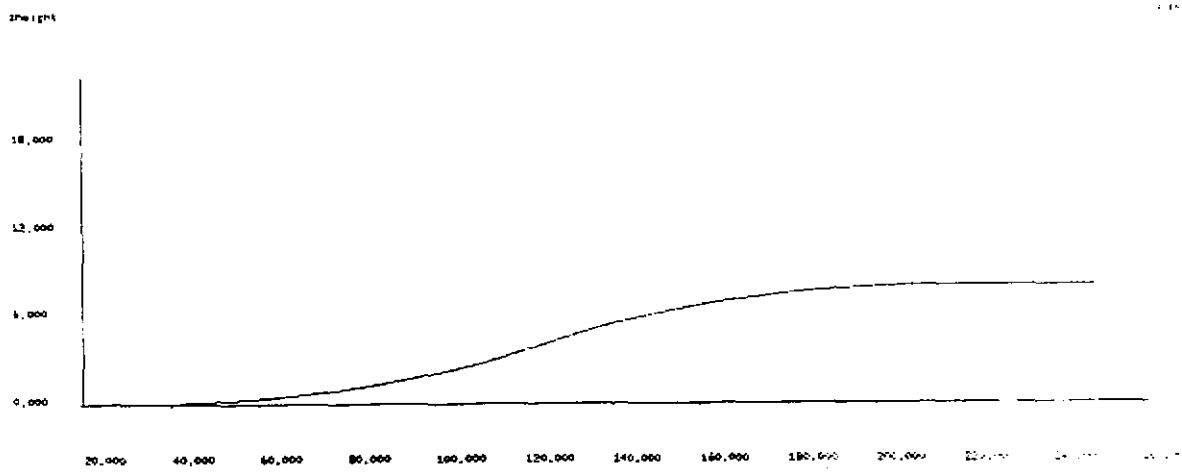

Figure 40: Road profile in global coordinates

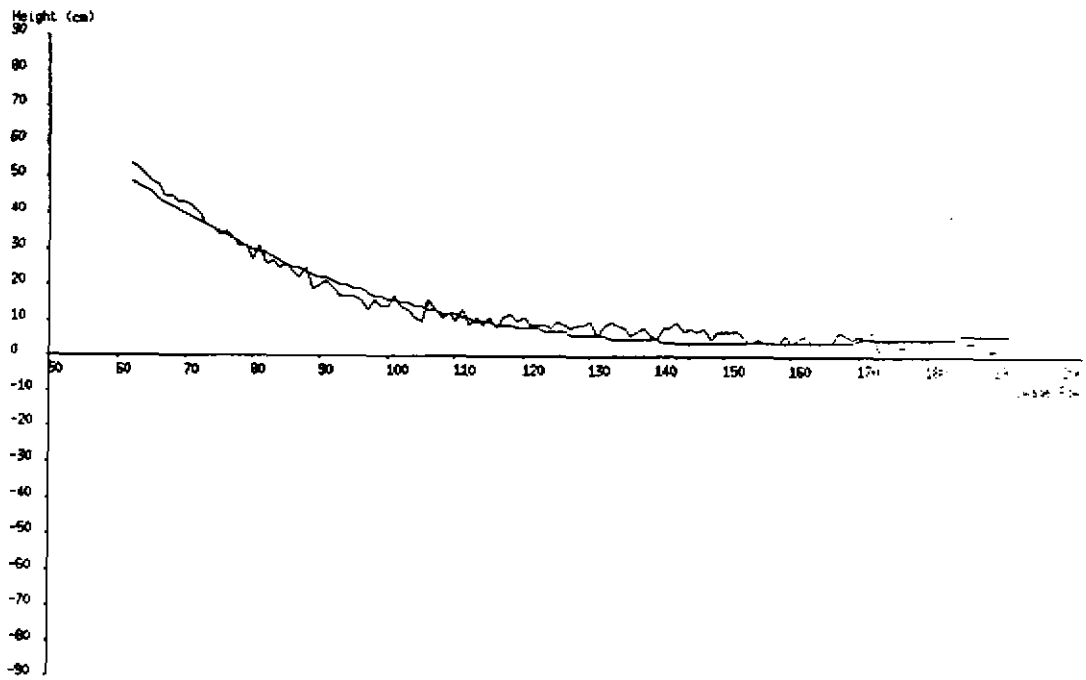

Figure 41: Fitted road profile in local coordinates

Figure 42 shows the results when an obstacle is present. Figure 43 highlights the obstacle

which was found in the middle of the road.

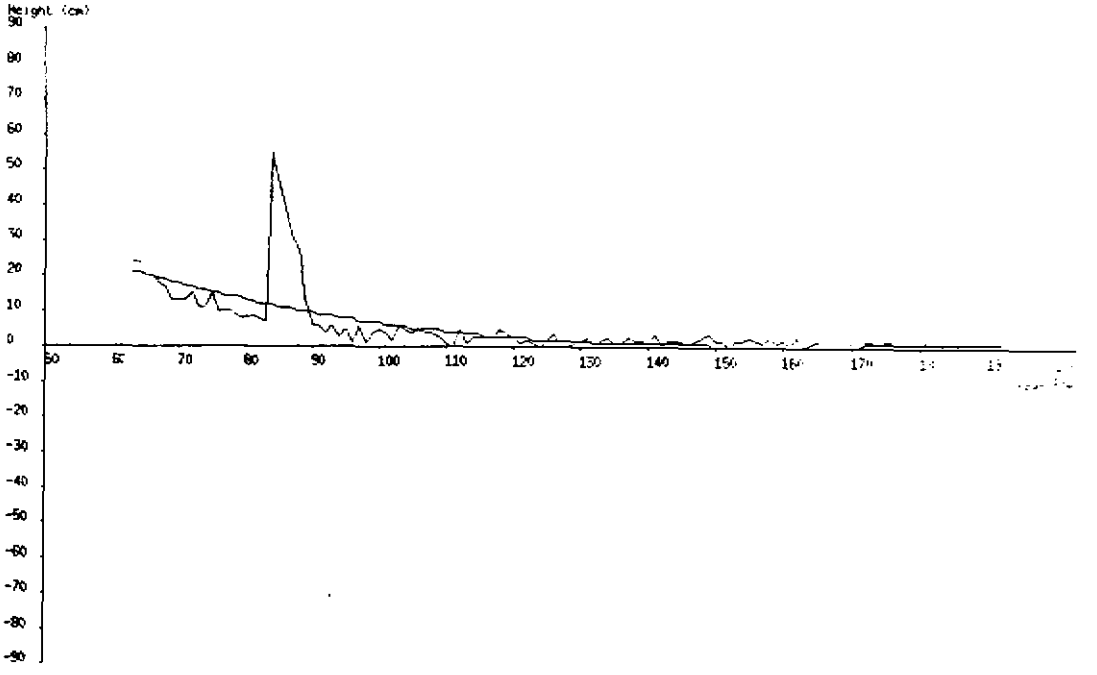

Figure 42: Fitted road profile in presence of obstacle

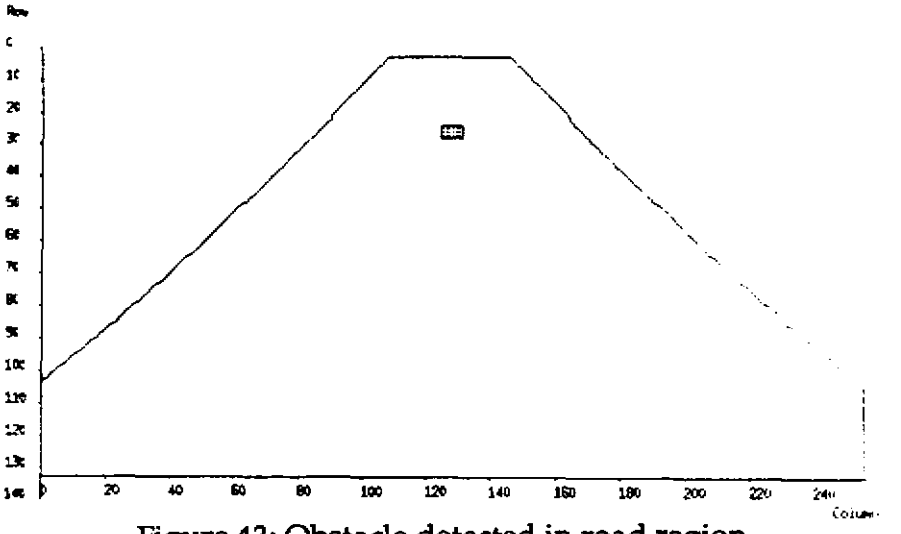

Figure 43: Obstacle detected in road region

There are a few cases where this algorithm fails. Road crown and bank cause problems due to lateral changes in the road height. Since the road height expectation is determined by the center of the path, whenever the heights of the edges differ significantly from that of the center then either spurious obstacles are found or valid obstacles missed. For example, the edges of a six meter wide road banked at 6 degrees are 26 centimeters above/below the center of the road.

Another problem arises when an obstacle lies in the center of the road either at the start of the scanner data or near the horizon. In these cases, the height model tends to follow the obstacle closely since there are no surrounding points to "pull" the fit back down. This results in the road model reflecting the height of the obstacle rather than the height of the road, as can be seen in Figure 44.

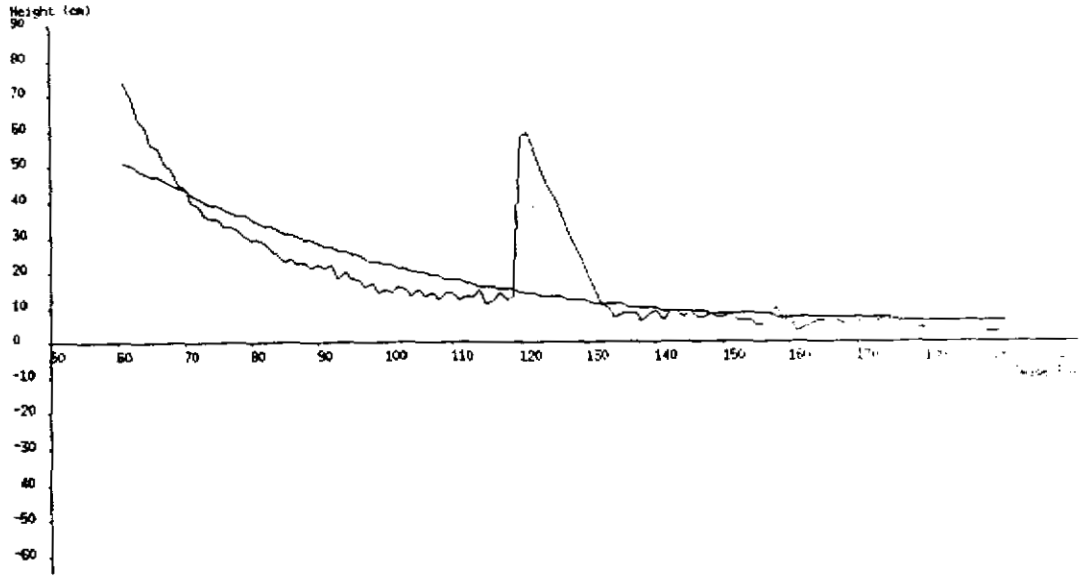

Figure 44: Fitted road profile in presence of an obstacle and steep grade

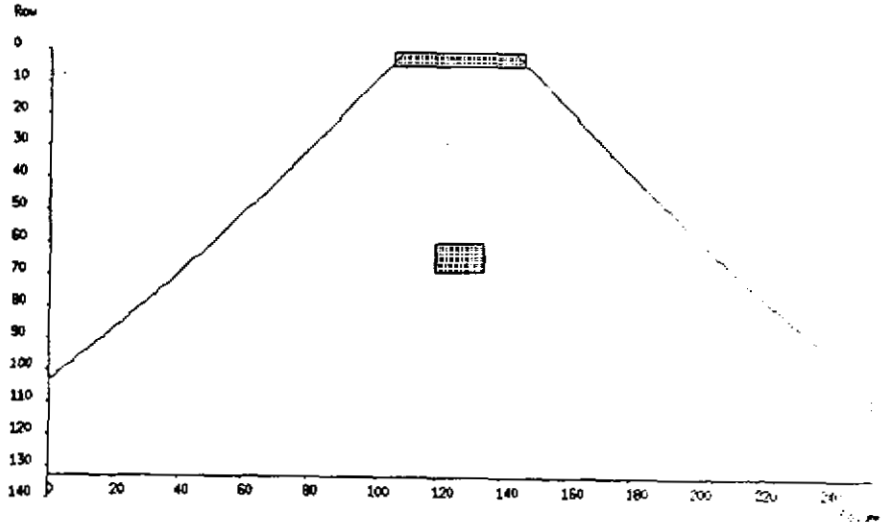

Figure 45: Erroneous obstacle found

There are just enough data points collected on the obstacle to prevent the fit from conforming to the steep hill. The result is that the algorithm is fooled into thinking that there is an obstacle at the far end of the road as shown in Figure 45. Note that although a bogus obstacle is found, the vehicle would still be slowed down and further processing would show that there is only one real obstacle.

3.3. Obstacle Detection- Conclusions

Our experiments using the flat world assumption proved to be insufficient because obstacles in realistic outdoor terrain could not be detected reliably. This is mainly because undulating terrain and vehicle pitch can cause either for obstacles to be missed or for false obstacles to be detected. Extending our methods to explicitly consider the road over which the vehicle is to travel made our algorithms less sensitive to the real-life problems mentioned above. Our final experiments with the NavLab and the Cyclone scanner were timed at 4m/s. We were able to detect obstacles in simulation of a scanner mounted on a vehicle traveling at 11m/s, given 256 x 256 range images.

Performance must be evaluated based on the fastest speed the vehicle is to travel and the smallest object that can be detected. Comparison of the methods presented above reveals that there is a trade-off between reliability and performance. Full frame analysis is clearly the best technique; however, both the sensing and computation necessary are prohibitively expensive.

Some caveats should be mentioned. It is possible that no obstacles are found in the collision zone due to occlusions by objects that are outside the collision zone. To guarantee a collision free path, a conservative estimate of the length of the collision zone must be made every time the path turns. Scanner frame rate becomes a factor at high speeds. When traveling around a bend, objects appear shifted in space so that some objects which are not on the path may sometimes appear to be on the path and vice versa.

A few improvements are suggested. In the selected scan approach, the grade of the road profile is calculated for each scan, and the grade on inclined roads is not predicted well. A similar effect is caused by small pitching motions of the vehicle. Both can result in a large difference in the grade being computed in successive scans. For thresholding to be effective, the grade of the road ahead of the vehicle should be fit using several scan lines, and then each line that is profiled should be checked against this fit.

Fitting a height to just the center of the road can lead to difficulties in extreme circumstances. This could be compensated for by using a weighted history, as in the selected scanline approach. Another method would be to fit a surface to the entire road. To reduce complexity, separate curves could be fit to the left edge, right edge and the center of the road. These lines could then be averaged, filtered or fit laterally to find the height for a given row in the image.

Benchmarks have shown that there is a bottleneck in the conversion from cylindrical range data to height data. This could be improved significantly by faster floating point hardware or by utilizing parallel processing. Bends in the path of the vehicle can cause parts of the path to be obscured from the field of view of a scanner rigidly mounted to the front of the vehicle. While this condition is detected and the only result is to force the vehicle to slow down, the scanner could be servoed to point towards the road ahead. Also, obstacle detection would be more robust if the scanner could be mechanically stabilized to negate the effects of vehicle pitch.

Regardless of which of the above algorithms is used, there are likely to be false alarms. The correct way to deal with these situations is to slow the vehicle and perform a more detailed and robust calculation in the region of interest.

4. Obstacle Avoidance

In the proposed scenario, the vehicle is brought to a halt as soon as an obstacle is detected in its path. Next, we would like the vehicle to navigate around the obstacle to the prespecified path. Since this capability is only invoked in an exception condition, it is acceptable to navigate around the obstacle(s) at a much lower speed than while tracking the path. We have designed a method that uses range information and vehicle position to navigate the vehicle. Optimal performance is replaced by the guaranteed satisfaction of a set of minimum requirements. These minimum requirements include decision-making (based on local feedback information in real-time) avoidance of obstacles, and convergence to a specified goal. The approach implemented is a modification of algorithms that are discussed in greater detail in [12].

We made the following assumptions:

- The obstacle environment is populated with obstacles that could be represented by convex polygons. If the obstacles cannot naturally be represented by convex polygons, an additional step is necessary to separate the obstacles into pieces that can be represented as convex polygons.
- The navigation algorithms only have access to global vehicle position and a description of the local environment in the form of a 2-D local map (built from a range profile) representing all the visible faces of the obstacle from the position of the vehicle.
- The vehicle is conventionally steered, having constraints on speed, acceleration, steering angle and the rate of change in the steering angle.

A careful examination of the problem indicated that it could be divided into two subproblems. First, given the final goal and the local map, a decision must be made as to whether there are any obstacles in the way and if so, which side of the obstacles the vehicle should go around. Subgoals must be selected such that the vehicle is led around the blocking obstacles. Second, once a subgoal is selected, a steering decision must be made which drives the vehicle toward a subgoal while steering it clear of the obstacles. To solve these two sub-problems, a *subgoal selection algorithm* (SSA) and a *steering control algorithm* (SCA) are proposed.

In the following section, we present methods used in the generation of local obstacle maps from scanner data. In subsequent sections the SSA and SCA are described. In the next section we discuss the implementation of the SSA-SCA combination on the NavLab and finally we present experimental results.

4.1. Extracting Features from Range Data

Our obstacle avoidance system used *Cyclone*, a laser range scanner capable of obtaining a single axis of range data at up to 8 scans/second (section 8.1.). The scanner was mounted on the front of the Navlab to collect obstacle information in front of it. The field of view was intentionally restricted to produce a field of view of 120 degrees to emulate the field of view of a scanner that was planned for the target implementation. The scanner data are processed to generate a polygonal representation of the obstacles within the scanner

range. During each sampling interval (typically 0.5 seconds), range data from the laser scanner is processed to produce a polygonal representation of the local obstacles. This feature extraction is performed in two steps.

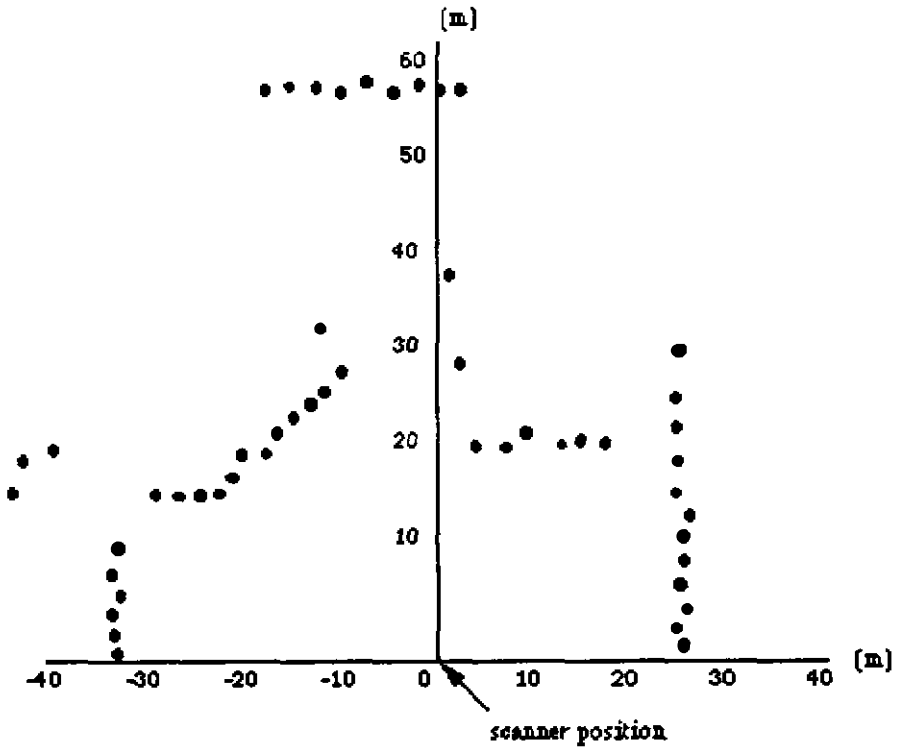

Figure 46: Sample raw data obtained from the range scanner

Figure 46 shows range data from one scan of the scanner in a hypothetical world. In the first step, range data is converted from polar to cartesian coordinates and the (x, y) points are clustered into contiguous blocks. That is, if two consecutive range measurements are separated by more than a preset threshold (s_1), a new cluster is started. Figure 47 shows a segmented form of these data. It should be noted that s_1 was chosen based on accuracy and desired level of detail required in segmentation. In any case it is not useful to set this threshold any lower than the accuracy or resolution of the range sensor. Cyclone has a resolution of 30 cm and we found that a value of 1m worked well on most terrains.

In the second pass, a common "polyline" algorithm is used to find a set of straight lines to fit each cluster [1]. This scheme uses a recursive method that ensures that the fitted lines are never more than a preset threshold s_2 away in lateral distance from the range data points. To start with, a single line is fitted between the first and the last line in the segment. For every point in each segment, the perpendicular distance to the fitted line is calculated. If this distance is larger than s_2, the procedure recursively picked the point farthest from the straight line (of distance m) making it a new breakpoint and replacing the current fitted line with two new line segments, until none of the points are more than

the threshold s_2 away from the fitted line.

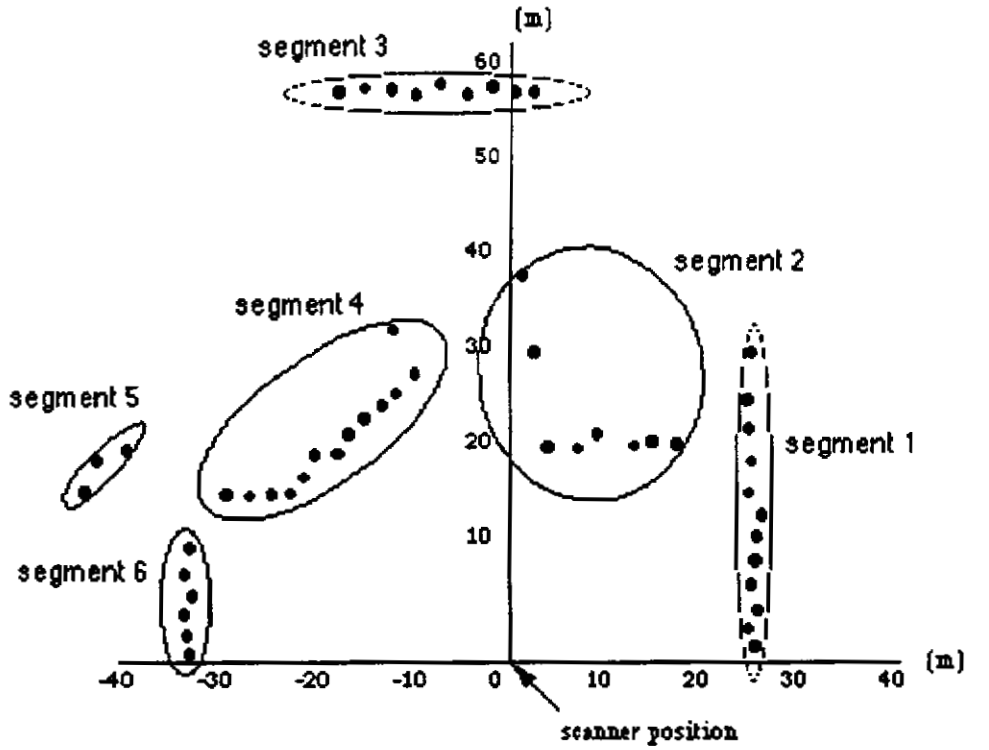

Figure 47: Segmented range data

This process is illustrated in Figure 48 (a), (b), (c) by fitting a polyline to segment 4 from Figure 47.

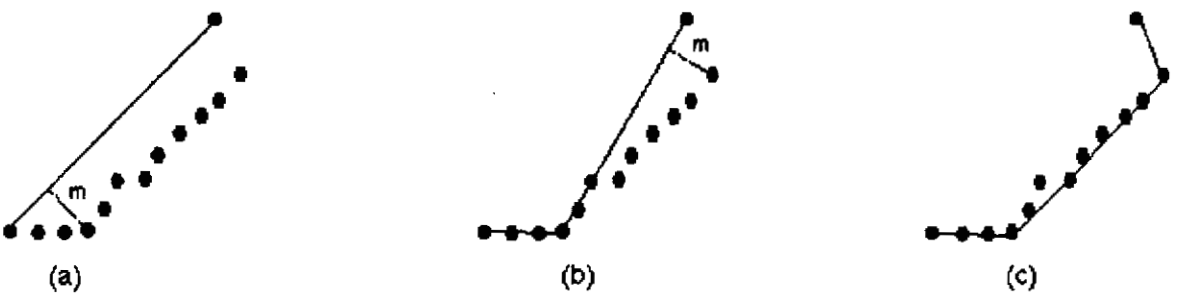

Figure 48: A polyline fit to segment 4

Relaxing the threshold, s_2, has the effect of a coarser fit with fewer lines to approximate the cluster. Tightening it has the effect of a finer fit, albeit with more line segments. Figure 49 shows the extracted lines obtained by application of the polyline algorithm to the data

from Figure 46.

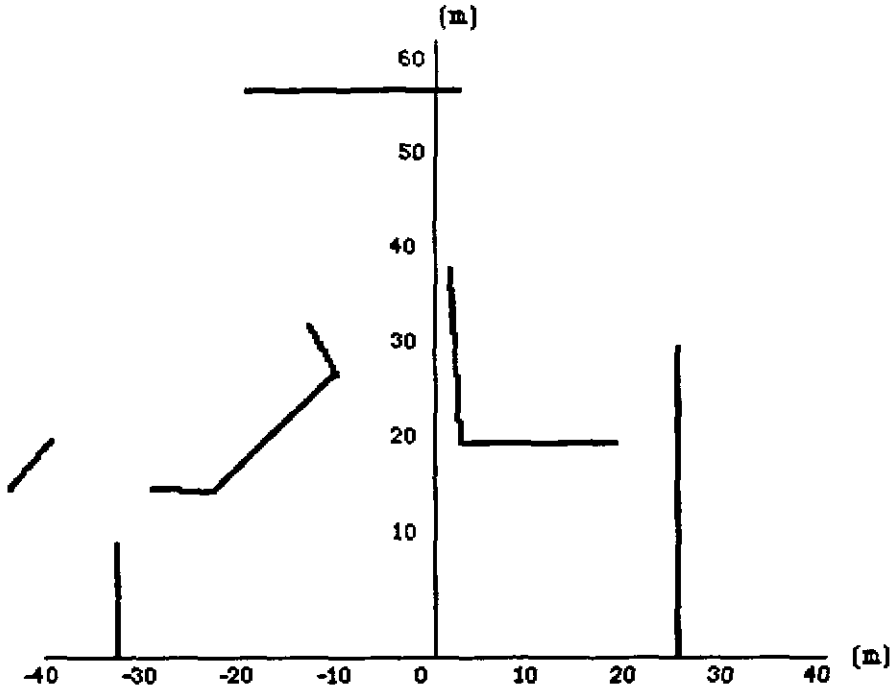

Figure 49: The polygonal shapes extracted from data of Figure 46

4.2. The Subgoal Selection Algorithm (SSA)

Often the final goal, the point to where the vehicle should converge, is not visible from the initial position of the NavLab. Subgoals are then required to guide NavLab around the obstacles until the final goal is visible. The subgoals are generated by the SSA. Each time the SSA is invoked, it first determines if the final goal is blocked by an obstacle. In determining whether an obstacle blocks a goal, we expand the obstacle by an amount equal to $r + \delta$ where r is the radius of the circle enscribing the NavLab and δ is a tolerance parameter. If the line of sight to the goal from the NavLab position intersects the expanded obstacle, we say the goal is blocked.

If the final goal is blocked by an obstacle, we marked this obstacle as an initial obstacle and another subgoal is generated to lead the NavLab around this initial obstacle. A subgoal could be placed on the right side or the left side of the obstacle. In the SSA we chose to go around the side of the obstacle that is closest to the final goal. The subgoal is thus generated next to the obstacle edge at a distance of $r + p$ ($p > \delta$). The parameter p is chosen to modify the characteristics of the NavLab trajectory, with larger p generally resulting in smoother trajectories. The subgoal is placed at the right (when subgoal direction is right) of the obstacle's right edge so that the lines from the obstacle edge to the subgoal and from the NavLab position to the subgoal are perpendicular to each other.

Once a subgoal for the initial obstacle is generated, it is checked to ensure that it is not blocked by, or too close to, another obstacle. New subgoals are recursively generated until the last subgoal is not blocked by nor too close to an obstacle. For the last subgoal, a *free*

space is generated. A free space is a triangular region which does not contain any obstacles. Two of the tree vertices of the triangle are the subgoal and the NavLab position. The third vertex is chosen to exclude the obstacles from the free space while allowing the free space to contain a large area for the NavLab to maneuver. We refer the reader to [11] for detailed rules for generating the free space.

Each subgoal is tested for *feasibility*. When a subgoal is not feasible, it means either the subgoal is outside of the region covered by the scanner, or the location of the subgoal requires the NavLab to change its trajectory in a way that is kinematically or dynamically impossible (e.g., to make a turn with a turning radius smaller than the minimum turning radius of the NavLab). The first case may happen if either the initial obstacle is already very close to the edge of the scanner's range or a cluster of closely bundled obstacles next to the initial obstacle extends all the way to the edge of the scanner's range. When such condition occur, we simply change the subgoal direction and try to generate a subgoal to lead the NavLab around the other way. When that also failed to give a feasible subgoal, the SSA returned the information that there is no feasible subgoal and the *default maneuver* is executed to bring the NavLab to a stop. A higher level program must then select a new final goal. On the other hand, if a feasible subgoal is selected, it is passed along with its free space to the SCA which determines the proper steering/drive commands.

4.3. The Steering Control Algorithm (SCA)

Given the subgoal and free space, the SCA uses a model of the NavLab to generate the steering/drive commands. The NavLab is a conventionally steered vehicle which can be modeled by the "bicycle" model with each pair of wheels collapsed into a single wheel (section 2.1.). The wheel base, denoted by l, is the distance between the axes of the front and rear wheels of the NavLab. The local coordinate frame for the NavLab is assigned at a distance of l_r from the rear wheel on the center line of the NavLab body. We refer to the velocity of the local coordinate frame on the NavLab as the NavLab velocity and the speed of NavLab at the rear wheel as the rear wheel speed, denoted by v and v_r respectively. The orientation of the NavLab, which is defined by the orientation, θ, of the local coordinate frame in the global frame, is critical in determining the future state of the NavLab.

For the purpose of generating steering/drive commands, we consider the NavLab as a point concentrated at the origin of the robot coordinate system. The finite size of the NavLab could be compensated for by means of expanding the obstacle regions in the scanner map. For the ease of generating a reference trajectory to be tracked by the NavLab servo, we choose the steering angle of the NavLab front wheel $\delta(t)$ and the speed of the rear wheel v_r (t) as two of the reference states and their derivatives as the steering vector.

The task of the SCA is to generate a piecewise constant steering vector at each sample interval. The steering vector is applied to the reference model to generate the steering/drive commands which results in a NavLab trajectory that avoids the obstacles. For the NavLab model, we have selected the angle of the steering wheel d and the speed of the rear wheel v_r as the state variables. For the ease of considering the obstacles when generating a reference trajectory, we choose two other state variables to be the x and y positions of the NavLab in the global coordinate frame, p_x, p_y. To specify a unique state of the

NavLab, the orientation of the NavLab, θ, is required as the fifth state variable. We note that for purpose of driving the NavLab only δ and v_r are required as the output of the NavLab model since these are the steering/drive commands that must be issued to the NavLab's servo controllers.

The overall structure of the SCA is illustrated in Figure 50. As shown, the steering vector $\mathbf{u}(t_n)$ is selected from a constraint set which is the intersection of the *system constraint* set $U_s(t_n)$ and the *environment constraint* set $U_e(t_n)$. The system constraint set $U_s(t_n)$ represents the kinematic and dynamic operating limits of the NavLab. When the steering vector is within this set it is guaranteed the resulting reference trajectory could be followed by the servoed NavLab within a known tolerance. The environmental constraint set $U_e(t_n)$ results from mapping the obstacle constraints into constraints on the steering vector using a representation of the free space, $C(t_n)$. When the steering vector $\mathbf{u}(t_n)$ is within this set, it is guaranteed that the state of the NavLab will be safe at the next decision time, $t_{(n+1)}$. Figure 50 indicates that if the constraint set $U_e \cap U_s$ is empty, the SDA generated a *default maneuver* which is a predetermined reference trajectory generated to bring the NavLab to a stop before running into an obstacle. When $U_e \cap U_s$ is not empty, the steering vector $\mathbf{u}(t_n)$ is chosen from $U_e \cap U_s$ to approximate a desired steering vector $\mathbf{d}^*(t_n)$ called the *objective vector*. There is great freedom in choosing the objective vector and performing the approximation. A more detailed discussion can be found in [13].

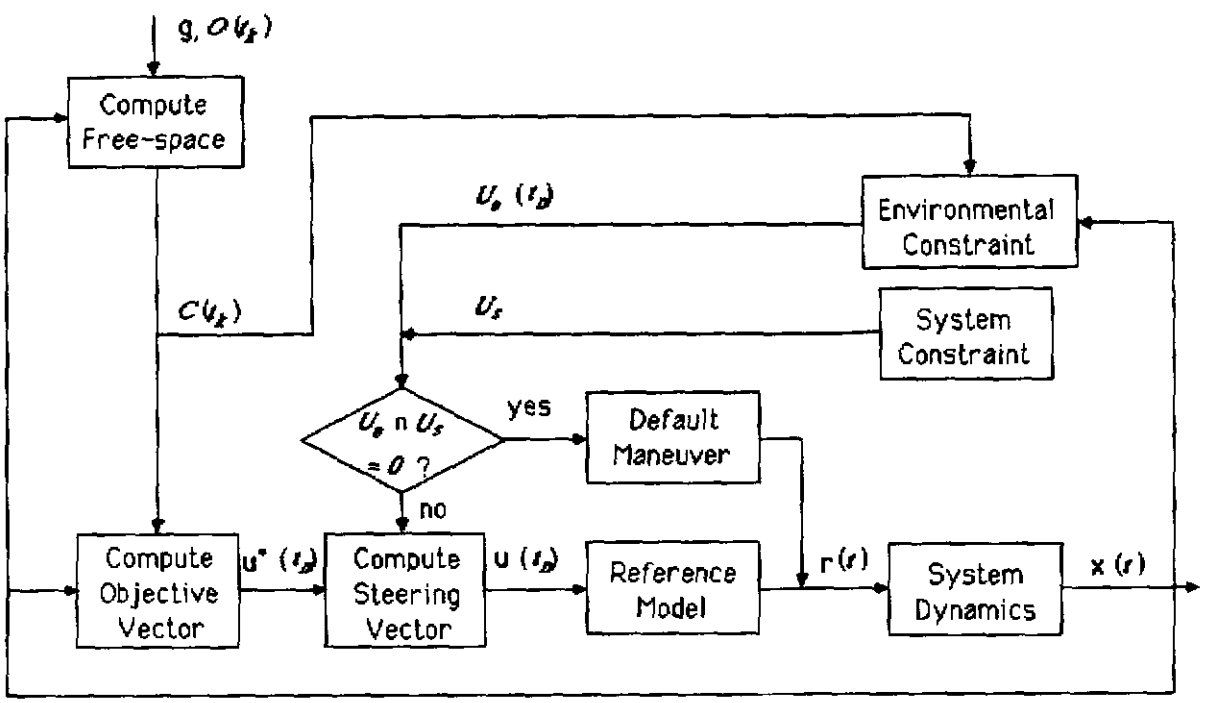

Figure 50: The feedback structure of the Steering Control Algorithm.

4.4. Implementation of the SSA and SCA

The SSA and SCA were implemented on a pseudo real-time system operating on a Sun 3 workstation. The scanner, SSA and SCA are synchronized through position tags available from the vehicle positioning system at a regular interval. The scanner is regulated by its

own servo controller (in this case at 2 hz) and a local obstacle map is generated every 0.5 seconds. The inertial guidance system sends a data structure containing the NavLab states including position, velocity and yaw rate to our obstacle avoidance program at 0.1 second intervals. Each NavLab state structure has a time stamp. The obstacle avoidance program checks each incoming state structure. At every 0.5 seconds, the SSA is invoked to generate new subgoals using the latest scanner map. At every 0.1 seconds, the SCA is invoked to generate new steering/drive commands.

The structure of the obstacle avoidance program is shown in Figure 51. The inputs to the obstacle avoidance program are the reference signal (final goal) from a higher level program, vehicle state feedback from the vehicle positioning system, and the local map from the cyclone scanner. Each time a stamped state feedback is available, it is checked to see if 0.5 seconds have elapsed since last time the SSA is executed. If so, a new local coordinate is established with the current NavLab position as the origin and the current NavLab heading as the positive x-axis direction. The location of the final goal is then transformed from the global coordinates to this new local coordinate system. The SSA is then invoked to generated new subgoals, using the local obstacle information generated by the PLA. Notice that local maps are naturally generated in the local coordinates. If no feasible subgoals could be generated by the SSA, emergency breaking, that is, the default maneuver is applied. The higher level program which generates the final goal is also notified to modify the final goal. When a feasible subgoal is generated, the SCA is executed to generated the steering vectors. The steering vector is integrated using the NavLab model to obtain the steering/drive commands which are sent to the NavLab servo controller. The SCA is executed at a 0.1 second interval.

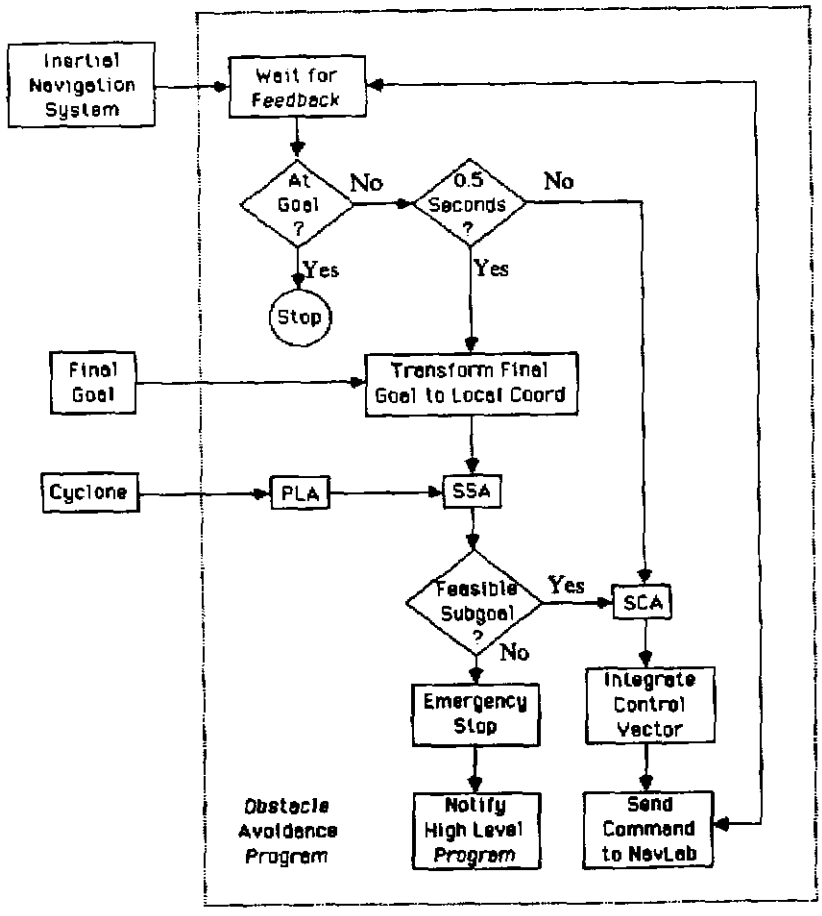

Figure 51: The structure of obstacle avoidance program for the NavLab.

4.5. Experimental Results

The obstacle avoidance algorithm was implemented in the C programming language in a Sun work station. This allows for easy development and porting of the program to the Sun computer on the NavLab. The program can also be run in simulation mode using a dynamic model of the NavLab. In Figure 52, a simulated obstacle environment and scanner data are shown in a global coordinate frame. The initial position of the NavLab is at (20, 10) location. The free space is shown as the triangular region with subgoal and

NavLab position as two of its vertices.

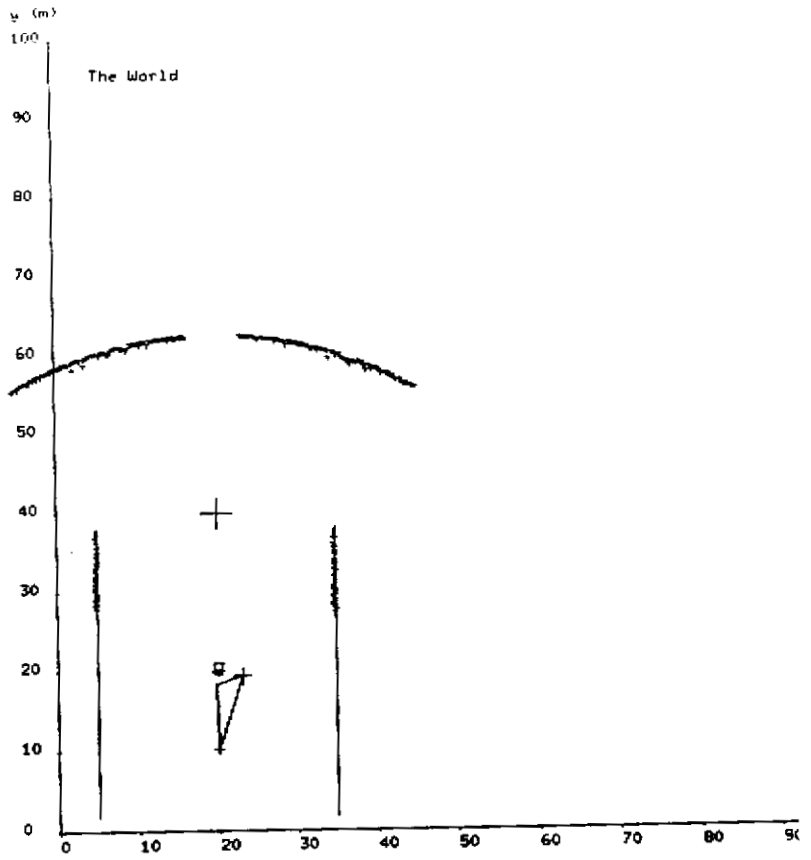

Figure 52: The simulated obstacle environment as viewed in global coordinate frame

Figure 53 shows the processed obstacle map and the free space (in the local coordinates)

Note the subgoal that has been generated.

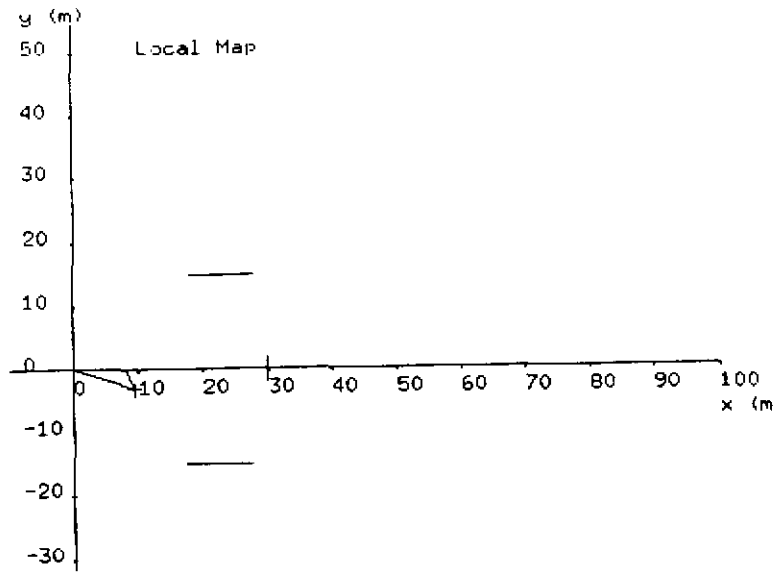

Figure 53: The simulated obstacle environment as viewed in local coordinate frame

Figure 54 shows the NavLab approaching a subgoal that is intended to lead the vehicle around the obstacle. Figure 55 shows the local coordinate representation of the scan.

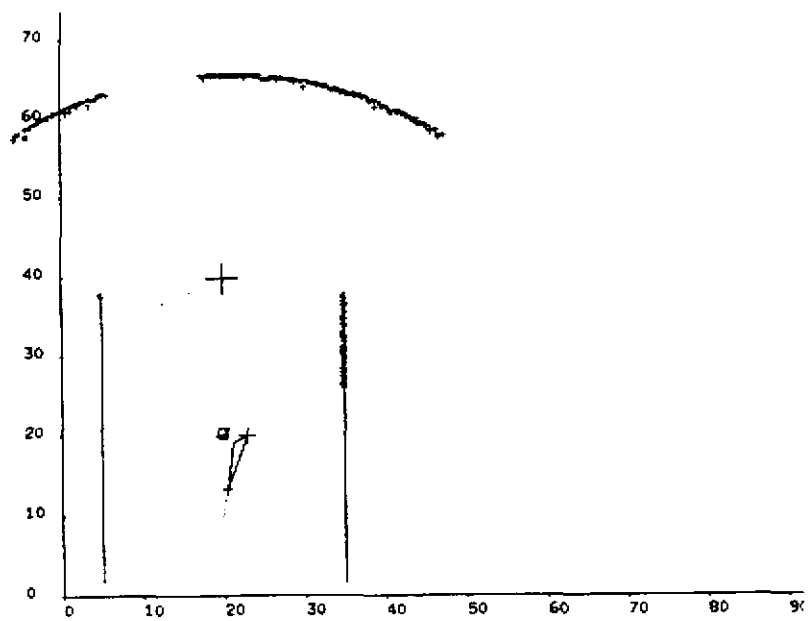

Figure 54: NavLab approaches a subgoal

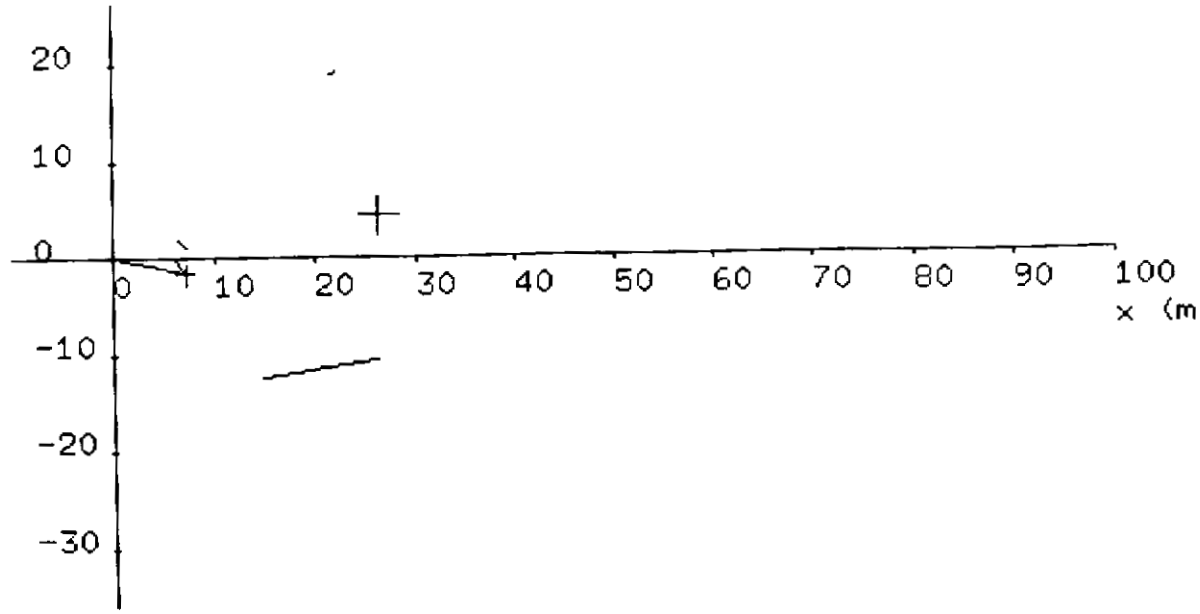

Figure 55: Local coordinate representation of scan from Figure 54

When the final goal becomes visible, new free space is generated as shown in Figure 56.

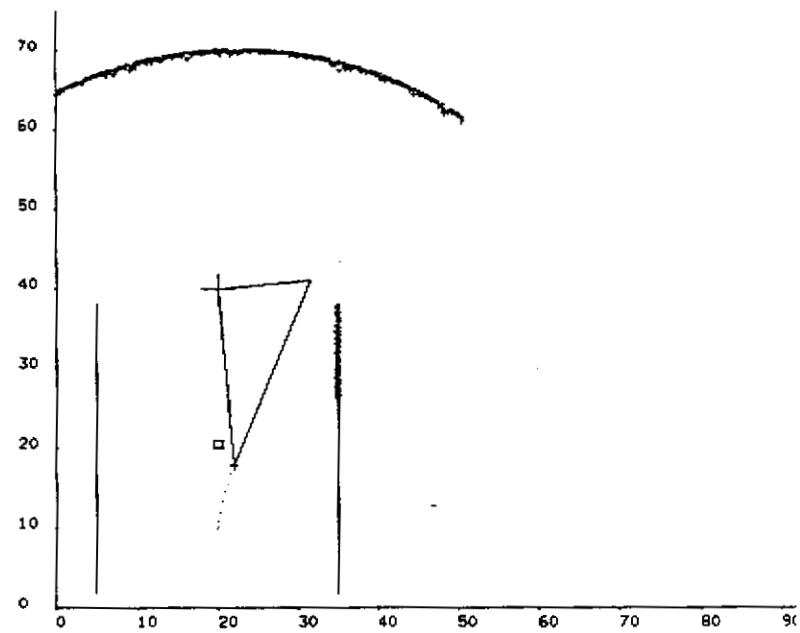

Figure 56: New free space is generated

Figure 57 shows the entire trajectory of the vehicle up to the final goal.

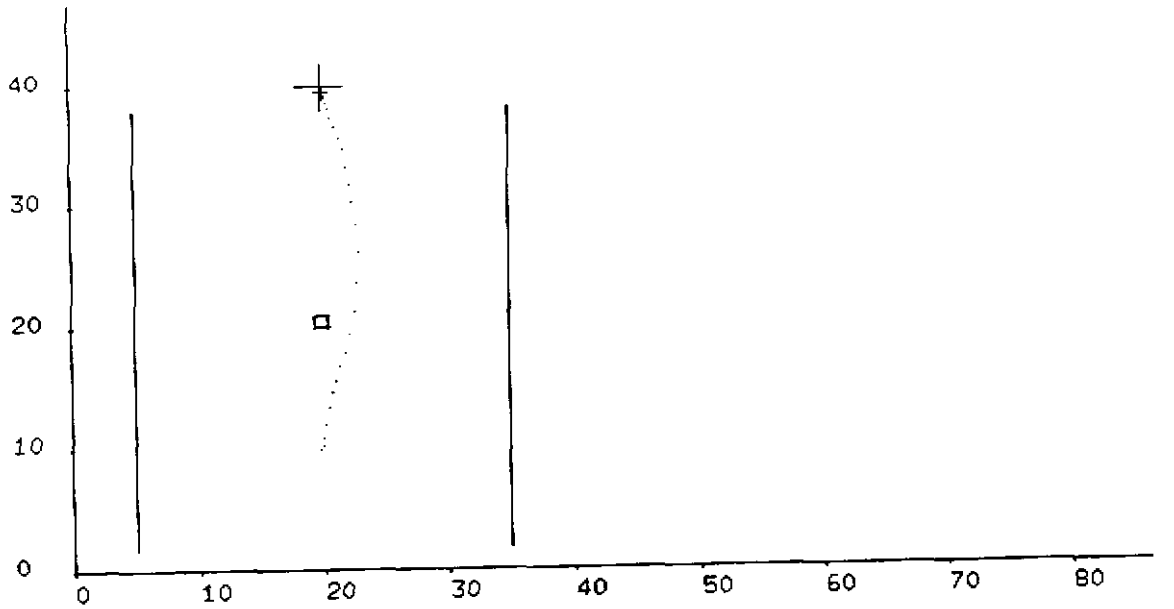

Figure 57: The entire trajectory up to the goal

Figure 58 shows the command signals issued to the vehicle during the course of the path shown above.

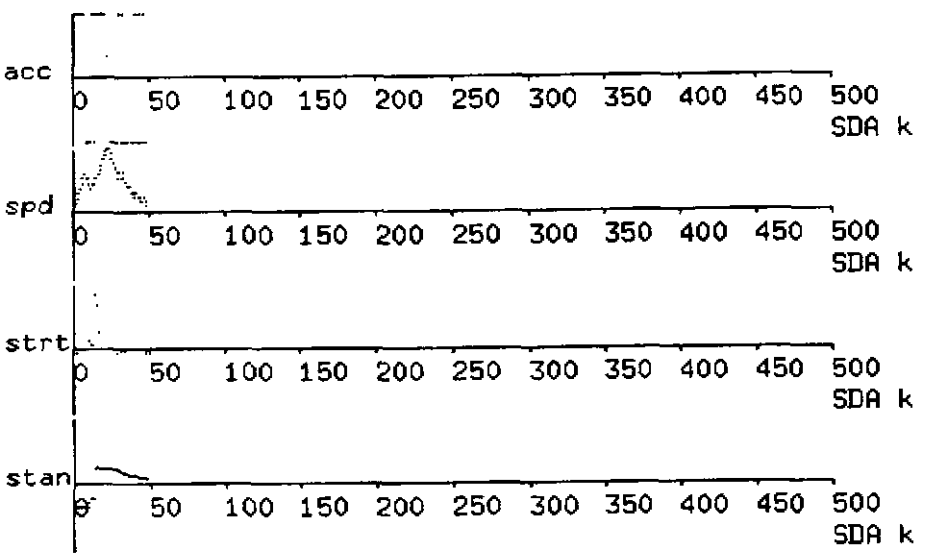

Figure 58: Command signals generated: **acc**- wheel acceleration, **spd**- rear wheel speed, ***strt***- *steering angle rate*, **stan**- steering angle

This simulation shows the obstacle avoidance algorithm is capable of generating smooth trajectories around the obstacles. In actual runs, the obstacle avoidance algorithm could drive the NavLab at a speed up to 2m/s. Above this speed strong oscillation occurs in part due to a delay in the control of the drive and steering of the NavLab, and in part due to

shortcomings of the algorithm, as discussed below.

4.6. Obstacle Avoidance- Conclusions

The final implementation was run on the NavLab through several obstacle courses. Results showed that the basic concepts behind the algorithm are sound. However, there were certain aspects of the algorithm that had to be modified to handle real life situations.

Consider first the scanner's limited field of view angle and its mounting position. The scanner is mounted in front of the vehicle, well ahead of the control point (by a distance of 2.7 meters). Also, the scanner is restricted to a fixed horizontal view angle of 90 degrees (45 degrees on each side of the longitudinal axis of the vehicle). This means that obstacles outside the horizontal view, on the sides of the vehicle couldn't be picked up by the scanner. This resulted in a serious problem when the program is used to drive the NavLab through a sharp turn, for example around the corner of a building, During approach, the corner disappeared from the scanner's field of view long before the vehicle clears the corner. Once the obstacle disappears from the local map, the vehicle tends to turn hard toward the desired direction and hence causing the vehicle to possibly run into an obstacle or at least to alternately oscillate towards and away from obstacles. This problem is inherent to the algorithm structure because it uses only the latest local map (in a effort to keep the computations for processing the map simple). To overcome this problem, it would be necessary to accumulate obstacle position information from local maps obtained from earlier iterations. Occupancy maps described in [3, 19] could conceivably be used as a method of incorporating data that is not immediately in view.

The other area in need of improvement is the detection and correction of infeasible subgoals. In the latest version of the program, once a infeasible subgoal is detected, the program is aborted without further examining the possibility of generating alternative subgoals. This could result in an program that is not very robust. For example, it is possible that noise in the range data could cause a single obstacle to be represented as a group of small obstacles located closely to each other. If a subgoal is chosen next to one of the obstacles in that cluster, it would result in a infeasible subgoal because the adjacent obstacles are so close. It may not be possible to come up with a strategy for choosing alternative subgoals in general. One viable way to solve this problem is to use heuristics that are most reasonable for the particular conditions encountered. For example, if a subgoal changes significantly, it might be assumed that noise is responsible for shift in the subgoal. In this case, a new scan could be used to recompute the new subgoal.

5. Architecture

A significant component of our work has been focused on the design of a computational architecture for a vehicle controller. This architecture is essentially the glue that holds together the various capabilities necessary to navigate a vehicle. Our design has been motivated mainly by considerations of generality and portability. We have deliberately developed abstractions in our architecture because it improves the design process and makes it easier to implement in a different computing environment. In short, the architecture is intended to:

- provide a means to interface sensors and actuators and human interaction into a real-time multiprocessing environment.
- provide an environment that aids development and trouble-shooting even at the cost of performance.
- lay out a framework that is independent of hardware implementation details like the type of processors used and the allocation of tasks on these processors.

In this section we present a discussion of the design principles we used in the design of our architecture and show the resulting specification for our architecture. We discuss special system utilities that we designed and briefly summarize the final implementation.

5.1. Design Principles

Through much iteration, we have arrived at some general principles to guide the design. They provide a consistent design methodology.

Separation of architecture and hardware: The architecture (except for the lowest level) is an abstraction of the hardware and software. No consideration is made about programming languages, processors, or specific communication protocols. This allows the implementation to be changed easily when necessary. In essence, the architecture is specified functionally.

Hierarchy of Architecture: Hierarchy in the specification of architecture is a design tool. The architecture is most abstract at the highest level and least abstract at the lowest level. Eventually the code that runs on the target system is a bit stream that does the "right thing" but for considerations of good software design, the architecture is specified in incrementally increasing level of detail. The same argument may be made in terms of architecture representation. Our experience is that choice of representation matters in the design process.

Distinction between control and data: Control is distinguished as information flow that causes action directly. Data is passive, it is information that is typically used in calculations that lead to control decisions. This distinction is useful because control flow can be shown in a state transition diagram reducing the system into a deterministic finite state machine. Control flow within the system is handled via messages. Data flow is handled via shared memory. Control and data flow to the outside world is done through I/O ports. Typically, control is asynchronous while data flow is synchronous.

Our architecture is comprised of a set of loosely couple "functions". Functions are a collection of tasks that are tightly coordinated to produce a particular capability such as the detection of obstacles. In the proposed architecture, the content of the functions changes with the application with the exception of *Supervision*. This function has a central role in the architecture in that it is responsible for initiating and suspending other tasks in the system. Once initiated, tasks continue on their own in a decentralized manner until an exception condition is signaled at which time the Supervision function initiates the appropriate steps.

Another function that is very useful is the *Navigator Interface*. This function is the focal point of all communication with the outside world and also serves as a secretary to the Supervision function. All requests and commands from the outside world enter through the Navigator Interface and are validated before they get any further. Instructions are passed on to the Supervision function, and requests for information are serviced directly.

Tasks are singular computational entities that are charged with performing a single task. For example, a task might have the job of performing I/O from an external device and another task would have the job of processing the input data. From the point of optimality, a singular program running on each processor is best but breaking each function into tasks allows for easier development and debugging.

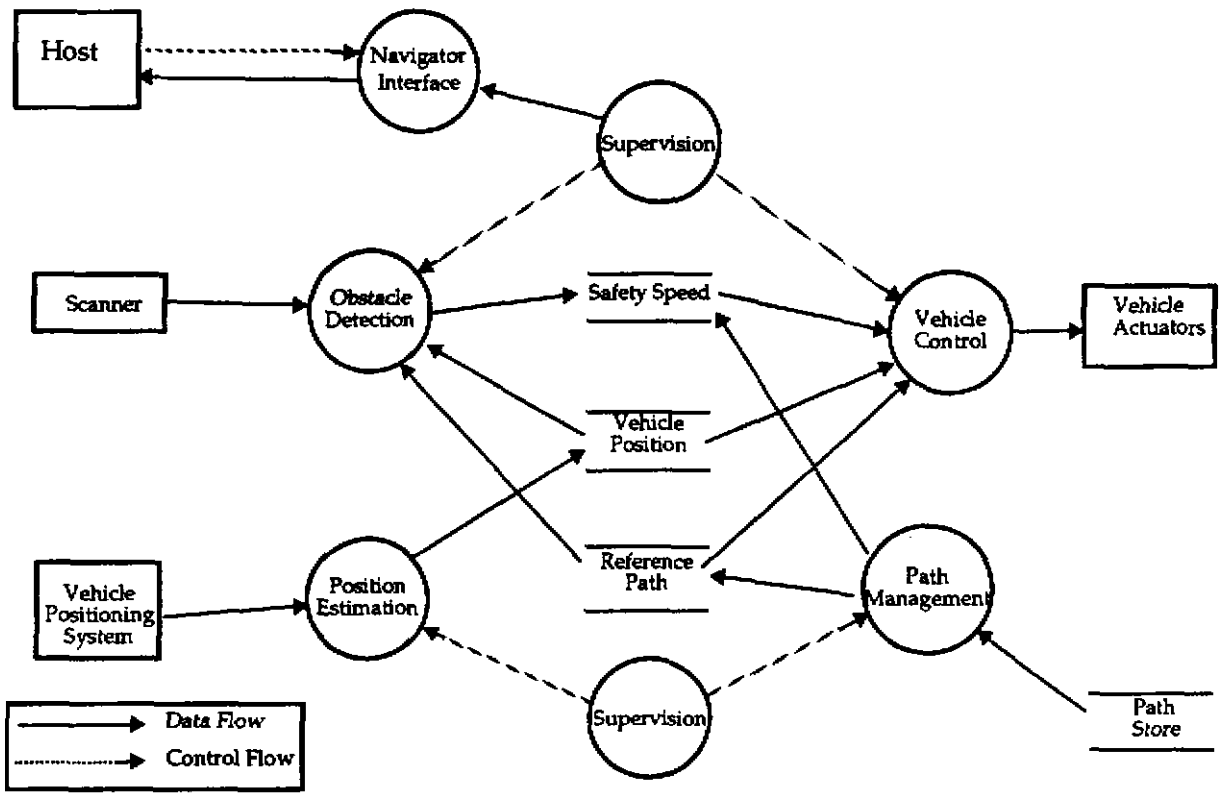

Figure 59: The FastNav architecture

Figure 59 shows the six major functions in the FastNav Navigator and their interaction with each other and with external devices. Our particular architecture has four specialized functions.

Obstacle Detection takes vehicle position, the projected path of the vehicle, and range data from the scanner and periodically posts a speed limit bases on proximity to obstacles. *Position Estimation* is dedicated to performing I/O to the Vehicle Positioning System (VPS) that periodically provides an estimate of vehicle position and velocity. This data is stored in memory to be used by other functions. *Path Management* takes path data from a secondary storage device and stores it in a ring buffer for other functions to use. This is mainly because the path may be long enough that it cannot be stored in memory all at one time. This function also periodically posts a speed limit that is based on the curvature of the future path of the vehicle.*Vehicle Control* is responsible for control of speed and steering. The path tracking algorithm described in section 2.2. is used to periodically compute a steering angle. The lowest of the speed limits posted by Path Management, Supervision and Obstacle Detection is sent to the vehicle.

We have also developed a consistent method of task interaction (as opposed to simply specifying the content of the information being exchanged) and of shared memory access. To this end we have developed 'primitives' for most commonly used procedures. For example, the primitive **closest_des_pos** returns the desired vehicle posture that is closest to the current vehicle position. This primitive can be used by various applications and is guaranteed to return the correct result. This approach is superior to having every application develop its own way of accessing the lowest level data structures to obtain required information.

5.1.1. Disparate Computing Environments

Along with establishing a real-time environment, we established a robust porting procedure, such that the work involved in porting applications from simulation to real-time is minimized. This was motivated by the fact that various project members worked in a variety of dissimilar environments. Further, methods and algorithms were constantly changing, and many changes affected other group members. For example there were two separate operating systems in use, each of which was optimized for different purposes – the Unix systems are optimized for development and ease of programming, whereas the Intel/RMX systems were optimized for performance.

Our work required four development environments:

- *Sun non-real-time*: This is the easiest environment to work in because it has excellent utilities, and most programs running in this environment are monolithic, which makes them relatively simple to debug.
- *Sun pseudo real-time*: This environment has the ability to provide data from sensors and send signals to actuators but without the guarantee of real-time performance. This environment is available on the NavLab but most importantly multiprocessing is slow because it is only loosely coupled over the ethernet. However, It proved very useful as an intermediate step to moving to real-time.
- *Intel simulation*: In this environment, we used the full-fledged capabilities of the real-time system but we simulated sensor both data and actuator commands. In this way, we were able to check the functioning of the real-time system independently of the real application.

- *Intel real-time*: In this environment, the real-time system was completely integrated with the sensors and actuators. Initially we used 80286 processors residing on a common back plane. The final system consisted of 3 80386 processors running the RMX286 operating system.

With the proliferation of environments we faced several troubling issues. Firstly, implementation of changing algorithms (common during the development process) in a real-time, multi-tasking, multi-processor environment required significant time and effort. Porting the developed code to a real-time environment required explicit knowledge of the application code. This led to redundant efforts. Even when a particular program was working in both environments (development and real-time), it was not possible to guarantee that both implementations performed identically.Thus our objective was to establish an environment in which it is easy to develop and debug algorithms as well as port developed code to the real-time system. To achieve this we adopted the following philosophies:

- *Modularity:* Modularity is sought between functionally distinct portions of the total system, making each module a "black box". The proper breakdown of the system into modules is crucial to simplifying overall system complexity and allowing direct porting of modules from one environment to another.
- *Communications restrictions*: Data communication is restricted to make data communication clearer and eliminate data conflict. Only communication through message passing and shared memory data structures is allowed and usage of global variables was kept to a minimum.
- *Export Procedures*: All system dependant details are hidden in primitives. The primitives are implemented differently for each environment, and thus applications could be written in a manner such that they were system-independent.

The resulting porting procedure is efficient in that code developed on a Sun workstation can be ported directly to an Intel system without any modification except for a flag that indicates the change in environment.

At the start of the project, we had intended to use the development environment (non-real-time) only to test fundamental concepts. After the basic concepts had been demonstrated, we intended to shift into a phase of developing programs on the target system that was meant to operate on the NavLab. This view turned out to be naive because of the iteration involved in the development of the algorithms. A year after we had started, we had two concurrent programming efforts- one in the Intel (target) environment, and the other in a Sun environment. The primitives helped immensely in porting the programs back and forth because we could hide implementation details of issues like message passing, arbitration, scheduling and shared memory access in the primitives while the programs could be identical at a high level. Further we found the Sun/Unix/X11 environment to be very useful in visualizing the enormous amount of data that is generated in simulations and experiments. This led to extensive usage of the Sun environment on the NavLab.

5.2. System Utilities

A separate effort was necessary to develop the lowest level of the architecture. This level must guarantee that the general assumptions made at higher levels are valid. At this level, the considerations are similar to those of operating systems. We have found it necessary to develop special utilities for the following:

Intertask Communication: A message passing scheme has been developed to pass messages between tasks running on multiple processors. The scheme allows specification of communication channels between tasks without explicitly specifying the hardware on which each task is running. Messages are used to

- invoke and suspend tasks
- relay information that arrives asynchronously
- relay important messages
- synchronize tasks

Most of the messages communicated are between individual tasks and the Supervision function which invokes and suspends tasks. Such messages are always acknowledged by the receiver. The responsibility lies with the sender to ensure that a message has been delivered. For example, if the Supervision function directs a task to switch a certain device, the task should acknowledge the communication after it has attempted to switch the device. If the task does not respond after repeated messages, a fault is inferred.

Synchronization: A single parent task on each processor is responsible for creating all tasks on that processor at power-up. After the tasks are created, they go into a state in which they wait to be initialized explicitly through messages from the Supervision function.

Some tasks need special synchronization apart from initialization and suspension. Often this is necessitated by concern for optimization and sometimes to ensure a sequence of actions. Such synchronization is handled by explicit messages. Note that these messages are somewhat like semaphores; but semaphores are not used because they are typically usable only among processes on the same processor.

Scheduling: Scheduling can either be explicit- tasks are orchestrated by other tasks or by events (like interrupts)- or can be implicit- the operating system allots time to each task based on user defined priorities. In either case, the following prioritizing must be maintained:

Task types	Priority
Safety considerations	High
Input from sensors and Output to actuators	High
Number crunching	Medium
I/O to external data stores	Low
Host interaction	Low

The effect is that tasks of the first two types run opportunistically on an as needed basis.

The last two also run opportunistically but only when the processor is idle. Such tasks can also be preempted by higher priority tasks.

Memory Arbitration: A deadlock situation can arise when two tasks on one processor are trying to access the same memory location. Across multiple processors there is the problem that one processor might alter the memory that another processor is currently reading. Further we would like to guarantee a consistent scheme of processor allocation which allows:

- multiple tasks to concurrently read the same memory
- tasks to be blocked from writing when any task is reading
- tasks to be blocked from reading while a memory element is being written to

Further, most data stores are ring buffered, so the arbitration protocol does not apply to the data but to the pointer to the head of the ring buffer.

5.3. Final System

We implemented programs to control the NavLab in both (Sun pseudo-real-time and Intel real-time). The results are as expected--at slow speeds the programs running on the Suns do just as well as the ones on the Intel; but, when greater computational load is on the hardware, the programs do not work well since they are not able to guarantee real-time response. For example, on the fastest run we have made so far (11m/s) it was not possible to plot inertial data on a display while the tracking program was running. The conclusion from these experiences is that both real-time and non-real-time environments are necessary. As much as it is possible the two environments should be similar and compatible. Ideally, a single environment would serve both purposes.

The final target system that we used consisted of three Intel 386 processors interfaced through multibus and using the RMX286 operating system. The 386 processors produced a factor of five speedup over the 286 processors that we had used earlier. The final archi-

tecture is shown in Figure 60.

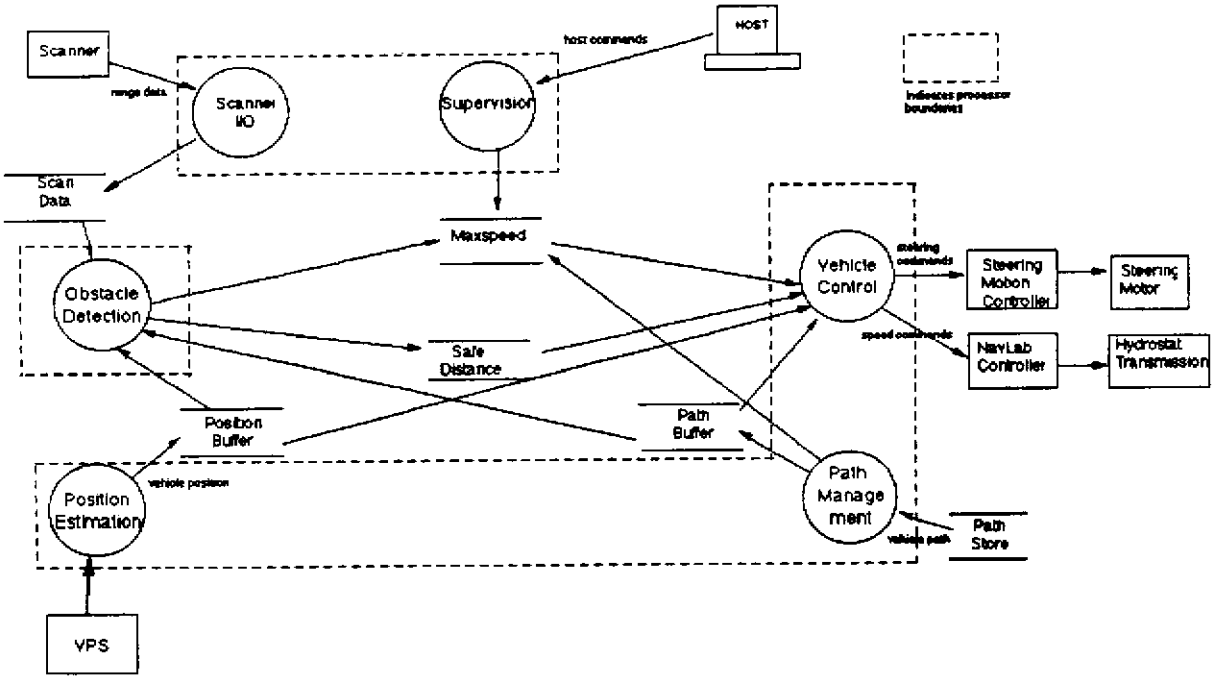

Figure 60: Final configuration of the FastNav controller

6. Fast Computing

We anticipated computing bottlenecks, especially for obstacle detection and avoidance. This is mainly due to the high bandwidth of range data that must be processed in a typical scenario. To deal with the high level of numeric computation, we chose to incorporate an array processor into our real-time architecture. Array processing is a special method for high speed processing of a set of identically structured data. The implementation presented here adopts an array processor specifically designed for high speed mathematical operations. The main advantage of adding an array processor into a multiprocessor system is that the main CPUs are free to perform other work while the array processor runs.

We implemented two of the most frequently called procedures (feature extraction from range profiles and coordinate frame transformation), on the array processor. Benchmarks show up to at 15 times speedup when the array processor was used. The importance of such a device is obvious from the following argument: we needed to process scans 6 to 8 times per second. However coordinate transformations and the polar to cartesian conversions alone took 100ms on a 80286 processor or roughly 80% of the time that could be spent on processing a scan. With the array processor (8 million instructions per second) the same computations take only 10% of the allotted processing time.

Later, however, when we upgraded the main computing engines to 386 based CPUs, the advantage gained by using the array processor was reduced because of the factor of automatic factor of 5 speedup obtained from the upgrade. It is also worth noting that with an array processor only a few routines can be sped up, while grading to a faster CPU results in a speedup of the entire program.

6.1. Hardware

We used an array processor which is specifically designed for high speed arithmetic operations through the use of a high speed, 16 x 16 bit hardware parallel multiplier and memory. Part of its efficiency comes from a microcode pipeline that can obtain the next instruction from program memory while the array processor is executing the current instruction. A flexible high level assembly language system, HIASM, is used for the software development.

The Array Processor Board (APB)[1] interfaces with the application system through a bus (multibus)as a memory mapped device. When the APB is integrated into the computing environment, it is transparent to the rest of the system. The host processor accesses the APB through a memory window which is set on the dual port memory of the on-board memory of the host processor. An interface software library is linked with the application program for controlling the APB. A block diagram of the Marinco array processor board is shown in Figure 61.

1. APB-3024M manufactured by Marinco, Inc.

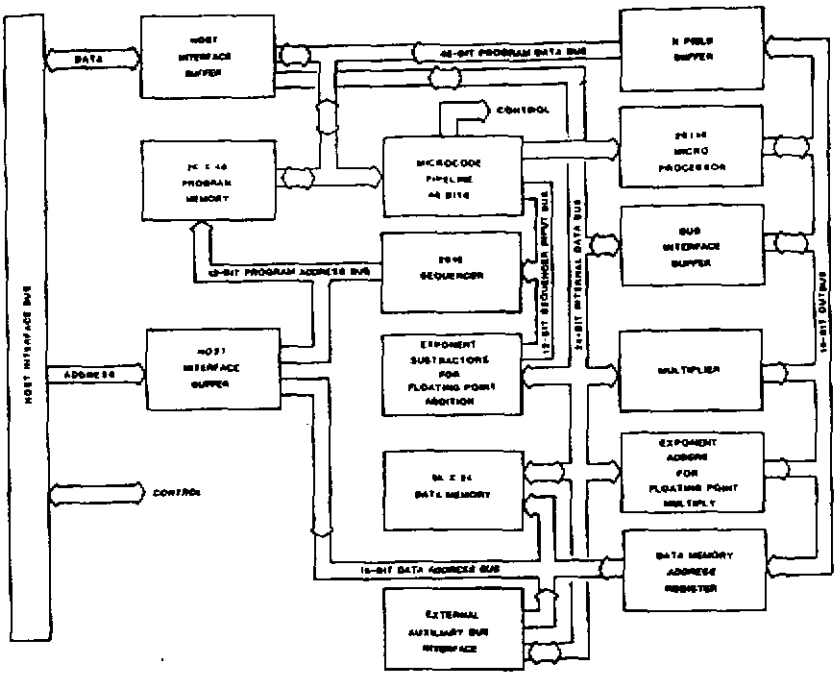

Figure 61: Hardware organization of the array processor board

The microcode pipeline register on the APB is a 48 bit storage device designed to hold the current microinstruction. Its primary purpose is to avoid lengthy access times to the program memory by supplying the microinstruction. The increased efficiency of the microprocessor is achieved by obtaining the next instruction from program memory while the microprocessor is executing the current instruction from the pipeline.

Both the program memory and the data memory can be accessed by either the host or the array processor. The host addresses both of them through the host interface bus. The array processor addresses the program memory through a microprogram controller. The microprogram controller accesses the program memory by placing the contents of its memory address on the program address bus. The array processor accesses the Data Memory by placing the contents of Data Memory Address Register on the Data Address Bus.

6.2. Development Environment

A reasonable software environment is essential to exploit the high performance of array processing hardware. A proprietary register oriented arithmetic language---High Level Assembler Language (HIASM) is used for software development. The HIASM language system combines the programming flexibility of a high level language with the machine intimacy of an assembly language. HIASM combines both program flow and processing instructions in one line of code. This offers the ability to branch to a different portion of the program and process information during the same clock cycle.

6.3. Applications

We have implemented several frequently used sections of code on the APB. Below we discuss two of the applications and present benchmark comparisons of computing speed.

6.3.1. Profile Generation

The polyline algorithm described in section 4.1. was implemented on the array processor as a typical application that is computationally expensive. Additionally, the order of the algorithm is very high in the worst case where the range data points are scattered in such a way that a large number of segments have to be fit.

6.3.2. Coordinate Transformation

There are two coordinate systems in our navigation system. The first coordinate system consists of world-fixed rectangular coordinates (x, y) in which the path is specified to the vehicle. Typically, the y-axis is pointed north and the x-axis is pointed east. Another coordinate system is based on a vehicle-fixed frame (v, w) in which the v-axis is pointed forward of the vehicle and the w-axis is pointed toward the left. The laser scanner used for obstacle detection is located at the origin of the vehicle coordinate frame.

In order to judge whether a collision will occur, the position of objects in front of a robot vehicle must be in the same coordinates as the intended path. The vehicle coordinates (v, w) transform to the world-fixed coordinates (x, y) by the coordinate transformation expressed in matrix form as follows:

$$[x\ y\ 1] = [v\ w\ 1] \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -dx & -dy & 1 \end{bmatrix} \times \begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ dx & dy & 1 \end{bmatrix} \tag{43}$$

where α is the angle between the v-axis and the x-axis and dx, dy is the location of the origin of the vehicle-fixed frame in rectangular (x, y) coordinates. The resulting execution times for the above two applications are presented in the next section.

6.4. Execution Speed Comparison

The profile generating and coordinate transformation algorithms were executed using the Intel 80286 processor (clock rate 12 MHz) and the Marinco APB-3024M array processor. The computed results using the 80286 and the APB-3024M are identical but the elapsed times are very different as shown in Table 1 and Table 2.

Table 1: Execution Speed comparison for profile generation

points	elapsed time on 80286/12 (ms)	elapsed time on APB (ms)	time ratio (28612/APB)
100	141.23	13.08	10.8
200	281.11	25.82	10.9
300	421.46	38.97	10.8
400	560.82	52.41	10.7
500	701.91	65.14	10.8

Table 2: Execution speed comparison for coordinate transformation

points	elapsed time of 80286/12 (ms)	elapsed time of APB (ms)	time ratio (28612/APB)
100	49.77	3.66	13.6
200	98.34	7.18	13.7
300	146.64	10.86	13.5
400	196.71	14.57	13.5
500	247.11	17.90	13.8

These comparisons indicate that the APB is about ten times faster than the 80286 processor for both the profile generation and the coordinate transformation. This is due to several reasons. Firstly, the Marinco APB is designed specifically to handle complex calculations, and most HIASM instructions can be executed in one clock cycle. On the other hand, each instruction on the 80286 processor can take between 2 and 20 clock cycles. Secondly, a powerful feature of the HIASM language system is the ability to write a statement that contains both an arithmetic instruction and a program flow command on one line of code. This feature allows the APB to process information and control program flow (for instance, branching) during the same clock cycle. Finally, the microcode pipeline register is able to obtain the next instruction from program memory while the microprocessor is executing the current instruction, so that the lengthy access times to program memory can be avoided and program running speed can be increased.

The cost of using an array processor is that special code must be generated for the routines. Also, there is a additional overhead in loading the array processor with data and shipping the results of the computations to the calling process.

7. Simulation of Range Data

Before the Cyclone Laser Scanner was operational, we needed some method of acquiring realistic range data. To this end we developed WB2, a 2D world modeling program. Arbitrary worlds could be built out of polygons and lines denoting obstacles and a vehicle path respectively. The program was then able to simulate range data obtained from a vehicle following a specified path among the obstacles. The data sets produced by this program were instrumental in debugging our obstacle detection and obstacle avoidance programs on the bench.

Another program (WB3) was designed and implemented to model 3D terrain and obtain data to simulate scanners that were not restricted to a single line scan pointed horizontal to the ground plane. The world model included road segments characterized by grade and bank. With WB3 it was possible to simulate various configurations of the scanner (number of scan rows and columns, field of view, scan timing, etc.) and the vehicle (speed and pitch) operating in a 3D terrain. WB3 produced frames of range data from a range scanner moving along a 3D path through a world of polygon-faced objects. Its input was a list of obstacles, a description of the path to be traveled by the autonomous vehicle, and a description of the scanner. A more complete description of the this simulator can be found in [25].

The main complication in the design of this simulator was that realistic shutter times (the amount of time taken to get a frame of range data) are significant given the speeds of the vehicle we wanted to consider. Thus, given a scanner that produces a range image at 2Hz and a vehicle that is traveling at 10 m/s, there is a difference of 5m in the starting and ending points of a single frame. This required a separate ray-cast for every pixel in the range image -- a process that made for very long execution times.

At first we had to run simulations of a vehicle following a fixed path at a predetermined speed to collect data and then apply the obstacle detection algorithms to the data after the fact. This was simply because the simulation took a very long time to run. Later, we were able to make the simulation of range scanning more efficient and were able to integrate vehicle and range data simulation in one program. This allowed for the range simulator to be used interactively with the vehicle simulation program. This worked particularly well when only a few scan lines out of the entire frame were being selected because we were able to simulate only the relevant scan lines, cutting down the simulation time by two orders of magnitude in some cases. This also allowed the obstacle detection program to be exercised on range data that varied with the speed and heading of the vehicle.

The simulator consists of two modules: a preprocessor (World Generator) and the Range Scanner Simulator program. The World Generator takes a simple description of the world (the road and the obstacles) and generates a representation of the world that can be used by a ray-casting algorithm. The range scanner simulator takes this representation and a description of the scanner and produces the range images. This process is shown in Figure 62.

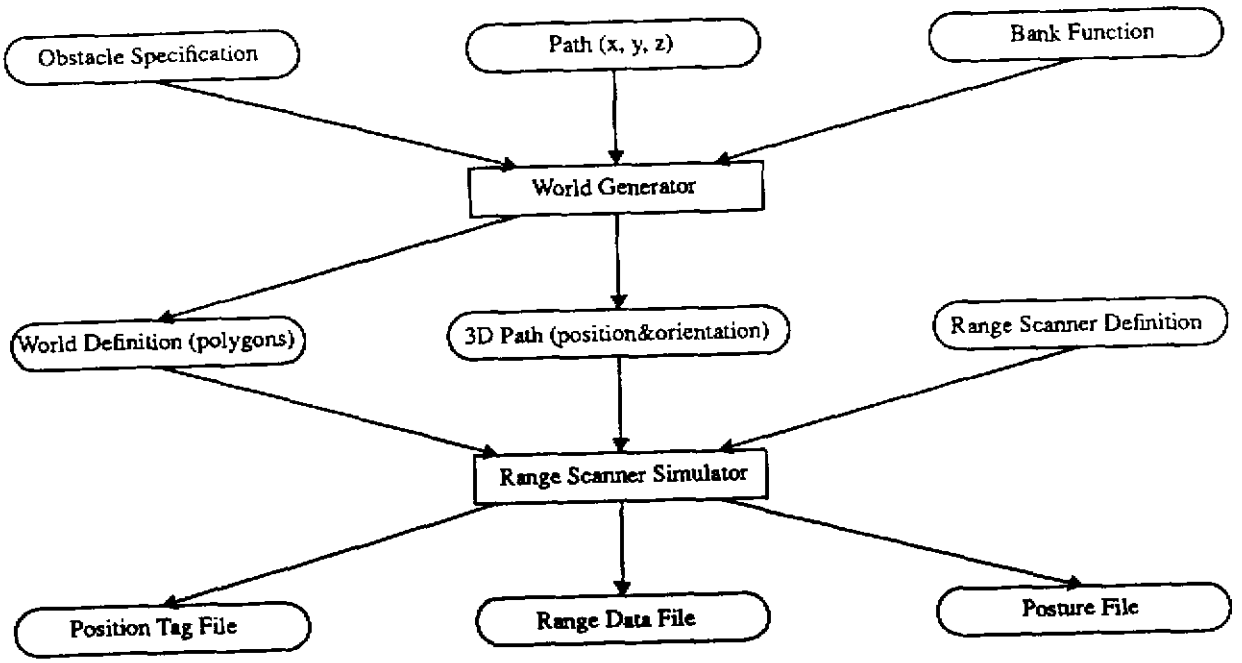

Figure 62: Functional block diagram of the range simulator

7.1. World Generator

The World Generator takes an *Obstacle Specification* file, a *Path* file, and *Bank Function* specification as input and generates a *World Definition* file and *3D Path Definition* file as output. The *3D Path Definition* file is the same as the original Path file plus some additional vehicle orientation information.

- Obstacle Specification file: Obstacles are chosen from a list of predefined geometric objects (cuboid, pyramid, triangular prism, wall). The location and dimensions of these obstacles are specified by a line in this file.
- Path file: The path file input to the World Generator is a file of x, y, z. These points specify the location of the vehicle along the path. The path can be considered to be one long polyline. The finer the resolution the better.
- Bank Function Specification: The x, y, z path specifies the location of the ve-

hicle and the forward direction of the vehicle along the path, but in order to complete the specification of the vehicle orientation at all points, some bank function must be assumed. Two methods were used: one comes from the relationship: tan(bank) = V^2/Rg, the other was to make bank the product of curvature and some constant. In both cases the radius of curvature was calculated from the 2D projection of the path onto the x-y plane.

- World Definition file: contains a list of polygons which contains (in any order) the road, the ground plane, and the obstacles.
- 3D Path Definition file: contains a list of vehicle positions and orientations.

In addition to the above two files, a third one that describes the scanner is used as input to the Range Scanner simulator. The *Range Scanner Definition* file contains details like the horizontal and vertical field of view, resolution and scan times, location and orientation of the scanner on the vehicle.

There are three parts to the world: the road, the obstacles and a ground plane. The ground plane is a large square, the obstacles are specified in the *Obstacle Specification* file, but the road is specified indirectly by the *Path*, the *Bank Function*, and a road width. The road is formed as a series of patches, where a patch is defined by five points. The five points are joined to make four triangles. In this way, the bank and grade of the road can be approximated. Figure 63 shows an example of a road patch.

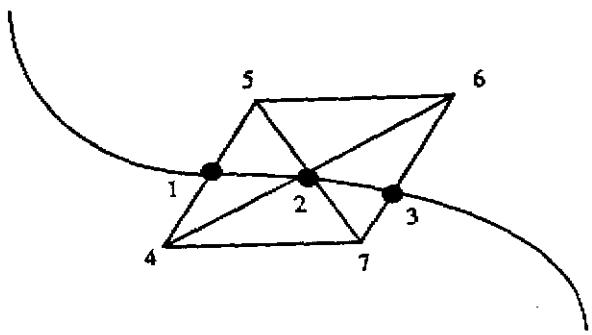

Figure 63: A sample road patch

A road path is formed by first sampling three points which are on the path, spaced at equal intervals (points 1, 2, and 3 in Figure 63). Then, points 4, 5, 6, and 7 are found using the width of the road. To form the road patch, the four triangles (2, 4, 5), (2, 4, 6), (2, 6, 7) and (2, 5, 7), are formed.

7.2. The Range Scanner Simulator

The Range Scanner simulator takes as input, the *World Definition* file, the *3D Path Definition file*, and the *Range Scanner Definition* file and produces a *Range Data file*, a *Position Tag* file, and a *Posture* file. The essence of this simulation can be captured by dividing the

process into three steps:

- interpolation of the vehicle position and orientation,
- transformation of the ray about to be cast by the scanner from scanner coordinates to world coordinates,
- casting the ray onto the world of polygons to find the nearest intersection.

These three steps occur for *every* range measurement, since it is assumed that the vehicle is moving at speeds comparable to the frame rate of the scanner. If the vehicle were assumed to be stationary for the entire frame, ray-casting could be replaced by the much faster approach of projecting the world of polygons onto the image plane of the scanner.

Since the vehicle path is represented by a polyline in 3D space, in order to find the vehicle position and orientation at an arbitrary distance, s, along the path, it is necessary to find which two vertices of the polyline the vehicle currently lies between and to interpolate x, y, z, i, j, and k. In this way, a smooth distortion of the scene due to changing vehicle position and orientation during the frame occurs, rather than a sharp, discontinuous distortion that would occur due to the vehicle passing a vertex of the polyline path in the middle of a frame.

The ray to be cast is calculated in two steps: the ray origin is found according to the current vehicle position and orientation and the scanner definition, and the ray direction vector is found according to the vehicle orientation, the current row and column within the frame, and the scanner definition.

Once the ray to be cast is found, it is cast onto the world of polygons. This involves analytically solving a system of three equations and three unknowns for points of intersection of the ray with each polygon in the world and finding the closest point of intersection, if any. The speed of the program decreases fairly linearly with the number of polygons in the world. On a Sun 3/260 with a floating point accelerator board, a simulation of 35 frames (each a 256 X 256 pixel image) in a world of 63 polygons took about 18 hours to run (about twice as fast as the same simulation using the 68881 math co-processor instead of the floating point accelerator board).

- Position Tag file: This file is generated by the Range Scanner Simulator to facilitate using the range data output of the simulator in simulations of the Obstacle Detection/Extraction and Collision Avoidance algorithms. This file was read in place of real VPS data.
- Range Data file: The Range Data file is the main output of the Range Scanner Simulator. It is a stream of integers which are the range values recorded by the sensor, in the order they are sensed and in the units of the scanner. For imaging purposes and sanity checks on the data, it can be converted to gray-scale images and displayed in sequence on a monitor.
- Posture file: At the beginning of the Range Scanner Simulation program (scan.c) the 3D Path Definition file is read into an array. Shortly afterwards, a Posture file (where "posture" is as defined in the Fastnav Primitives document) is output to a file. This file was used in later simulations of the Obstacle Detection/Extraction algorithms in order to simulate "perfect path

tracking" (no position or heading error).

7.3. Range Simulation- Conclusions

Figure 64 shows an example of a simulated world consisting of a road and polyhedral objects. A range scanner with a field of view of 60 degrees (azimuth) by 90 degrees (vertical angle), mounted on a vehicle traversing through this terrain was simulated. Figure 65 shows one of the simulated range images. Note that there is very little contrast in the image because there are few range discontinuities.

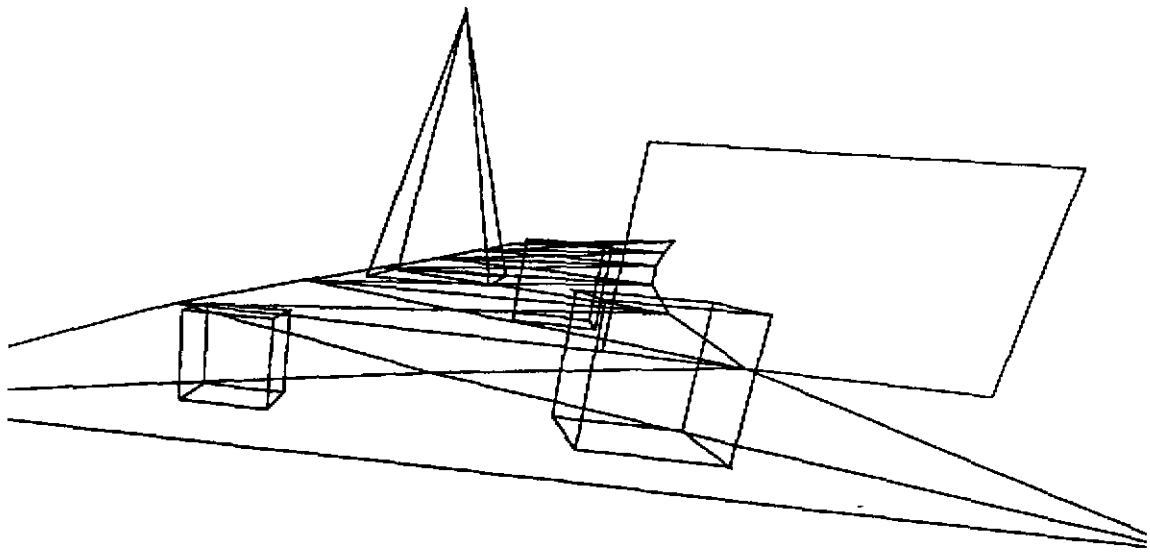

Figure 64: An artificial world consisting of road "patches" and polyhedral objects

The best feature of this simulator is the generality in that a large family of scanners can be modeled following an arbitrary 3D path. The worst feature is the speed – the method of ray-casting is slow. Using more macros instead of function calls might significantly speed up the process, or maybe a whole new method of ray-casting (such as an octree-based implementation) is called for. Another shortcoming is the fact that road crown is not modeled at all. A crude approximation to crown (a crease down the middle of the road) could be achieved by breaking each patch of the road into eight triangles instead of four. Of course this would approximately double the number of polygons in the world and therefore double the run-time of the simulation.

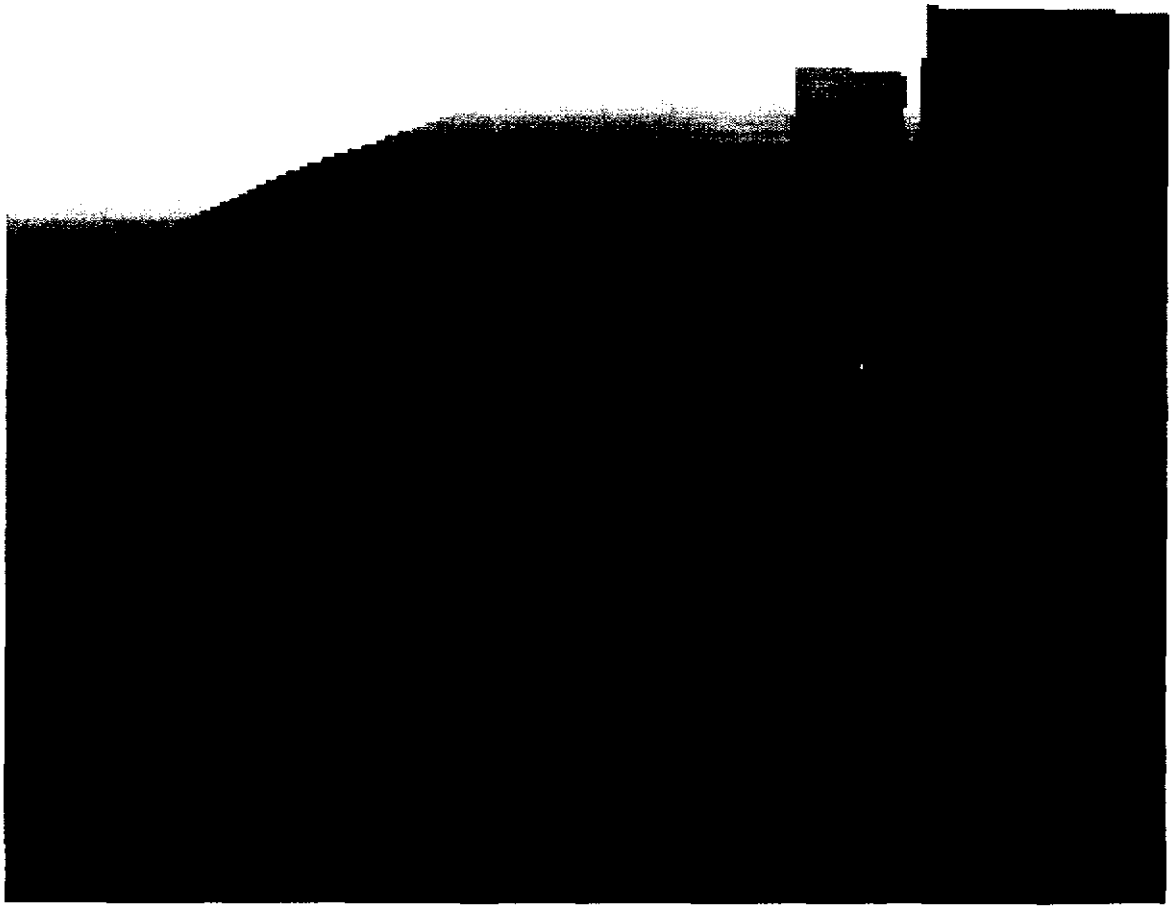

Figure 65: Simulated range image while traversing terrain shown in Figure 64

Despite these deficiencies, the simulator served a useful function. A slight variation of it was incorporated into the simulation version of the Fastnav Primitives in the Sun environment, facilitating development and testing of the Obstacle Detection/Extraction and Collision Avoidance algorithms before they were tested on-board the Navlab.

The simulator proved to be particularly useful in investigating the problem with the unexplained displacement of obstacles viewed from a range scanner during simulations of a vehicle traveling at a high speed using a slow (but realistic) frame rate. We were precisely able to pinpoint the cause of this phenomena because we were able to control the simulations very precisely and deterministically.

8. Devices

During the course of our research we have had to build two devices in-house. Cyclone, is a scanning laser rangefinder that was used for obstacle detection. NavArm is a test device built to test the obstacle detection algorithms. We have also had to incorporate a major system for providing vehicle positioning at a high rate.

8.1. Cyclone Laser Scanner

Range information is essential for robot vehicle navigation. Even if a robot's path has been preplanned, in a realistic scenario, the robot must perceive obstructions that might interfere with its travel. We have designed and implemented a scanning rangefinder, *Cyclone*, that enables a robot vehicle to determine the three dimensional shape of its environment with high accuracy. The device, built over a period of 18 months starting in mid-1987, has since been used in various configurations for obstacle detection, collision avoidance and In the following sections, we discuss the motivation for such a device, its design and some of its applications. A more detailed discussion of the design of the scanner can be found in [30].

8.1.1. Background and Specification

Having chosen ranging as the means to accomplish obstacle detection and collision avoidance, it remained to choose from the many methods possible to obtain range information.

Recently, some devices have become commercially available that allow the measurement of the time of flight of laser pulses. These devices, designed for surveying applications, have been demonstrated to measure distances up to 5 km using retro-reflectors. For natural scenes, the range is much shorter, between 50 and 100m. There is a trade-off between the maximum range possible and the dynamic range (the difference between the maximum and the minimum range that can be measured). If it is necessary to make measurements over a large interval, the maximum range must be compromised.

Our primary application, obstacle detection, dictated the functional specifications for the sensor, the scanning mechanism and the electronics:

- Range measurements up to 40 m (maximum) and 3 m (minimum)
- Maximum scan rate between 10-15 hz
- Narrow vertical field of view (between 1-3 lines of range data)
- Angular resolution of less than 1 degree
- Robust enough to withstand extended use (4-6 hours at a time) in natural environments
- Immune to high frequency vibrations from vehicle
- Immune to electromagnetic noise in the environment
- Able to relay data in real time

- Flexible enough so that scanning speeds and angular resolution can be changed easily.
- Cost not to exceed $30,000

The dynamic range specifications and cost limitations led us to choose the time-of-flight instrument mentioned above.

The other main decision was that of scanner configuration, the mechanical system in which the laser is housed and which determines the distribution of the laser beam emanating from the range finder. Since the objective is to recognize obstacles on an otherwise flat road, we didn't think it necessary to use a dual axis range scanner (with a nodding as well as rotating mirror) that typically provides from 80 to 256 lines of range data per frame. Most of these devices are slow, providing 2 frames per second at the most. Rather, what we wanted was less data, but more often. It was estimated that 10 scan lines/second from a single line scanner would be necessary for obstacle detection at speeds of 10 m/s.

8.1.2. Scanner Design

We considered two configurations for the scanner to distribute a laser beam from a single point source. The first one involved rotation of the entire rangefinding device. The advantage of such a configuration is that laser energy transmitted and received is not attenuated by mirrors or prisms. However there are several complications. It is necessary to use a large sophisticated slip-ring to transfer data and power to the instrument. Further, spinning a mass of several pounds brings risks stability, and increases vulnerability to natural elements and accidents.

The second configuration we considered used a rotating mirror to point the beam. This involved rotation of a tube containing a mirror (with an adjustable set angle) above the rangefinder aperture. The advantage of such a method is that the rangefinder itself is not required to move, making a safer and simpler mechanism. This design was chosen and implemented. Designed to be mounted on the front of an autonomous vehicle, the system consists of several parts:

- Scanner enclosure: houses the rangefinding unit, a DC motor/encoder, rotating tower and an interface circuit
- Power and communications enclosure (PCE): provides the power and control signals for the scanner motor. It also contains circuitry to buffer range data from the rangefinder and to relay these data to a host computer.
- Host computer: connected to the PCE by a parallel data interface to read buffered range data. Apart from processing the range data, it is also responsible for sending high level control commands (motor speed, etc.) to the PCE. These are then turned into low level controls sent to the motor.

These modules are shown in Figure 66.

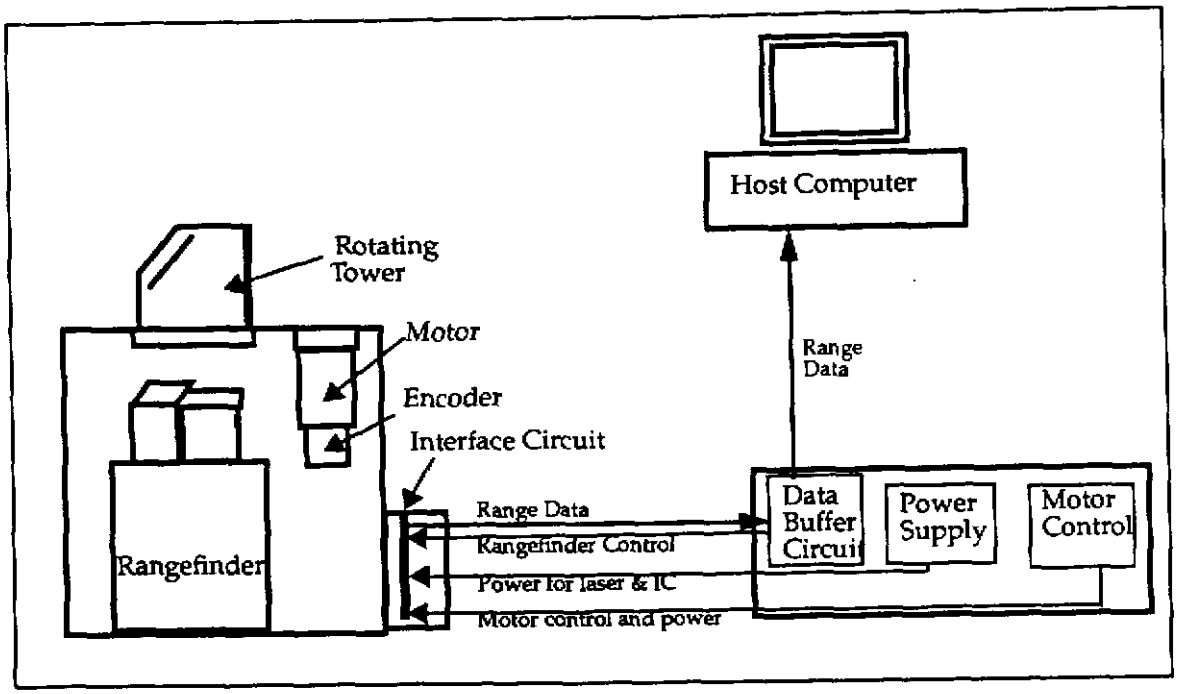

Figure 66: Configuration of the system

8.1.2.1. The Rangefinder

Cyclone uses a customized version of a commercially available single point laser ranging device (RF-90) [1]. Salient features of the sensor are:

Wavelength	0.9 micrometer
Laser	Gallium Arsenide Semiconductor
Pulse repetition rate	7200 Hz
Range	3 - 50 m
Accuracy	+/- 15 cm
Beam spread	2.5 mrad
Resolution	10 cm
Power requirements	12 V DC, 1A.

The instrument is able to time the flight of laser pulses transmitted and returned from naturally diffuse objects. The device has been modified such that rather than being triggered by hand, the laser is activated by a pulse train. Figure 67 shows a block diagram of the operation of the rangefinder.

1. Built by RadarTechnik and Elektrooptik, Austria.

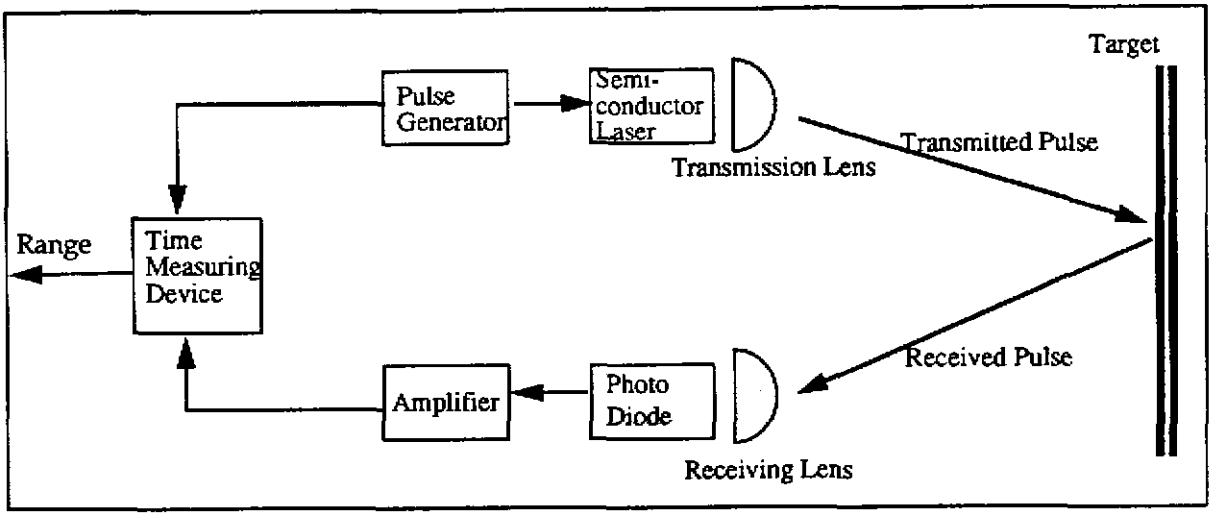

Figure 67: Operation of rangefinding device

Other researchers have done additional work on calibrating the sensor and have found that it is possible to reduce range error, especially for measurements taken from distances of less 3 m [23, 24].

Interfacing with the RangeFinder

When the rangefinder is triggered, the measured range is output on an 11 bit parallel port. Specifically, two control lines must be manipulated for the rangefinder to take a measurement:

- TRIGGER is a control line that, when given a rising edge, causes the rangefinder to produce a range measurement.
- LASER_INHIBIT is a control line that must be driven low for the TRIGGER inputs to be effective. We have used this line for safety purposes by ensuring that this line is never pulled low unless the tower is rotating.

Additionally, two status lines are available:

- DATA_VALID is a status line. It is held high only when a good echo (reflected signal) is received by the rangefinder.
- DATA_READY is a status line which is held high when the data are ready to be read. This line is always high except for 20 microseconds while the data are in transition.

Reading data from rangefinder can be accomplished by the following algorithm. A count down timer is started after a TRIGGER pulse is issued. Immediately, a loop is initiated that checks to see if the DATA_VALID line has been pulled high. If this doesn't happen by the time the timer completes, the data are considered to be "missing". If the DATA_VALID line has been pulled high, then the algorithm waits for the DATA_READY line to be pulled high. As soon as this is done, the parallel port can be read.

8.1.2.2. Scanner Enclosure

The scanner enclosure houses the rangefinder as well as the necessary optics and electronics. Figure 4 shows a cut-away view of the scanner.

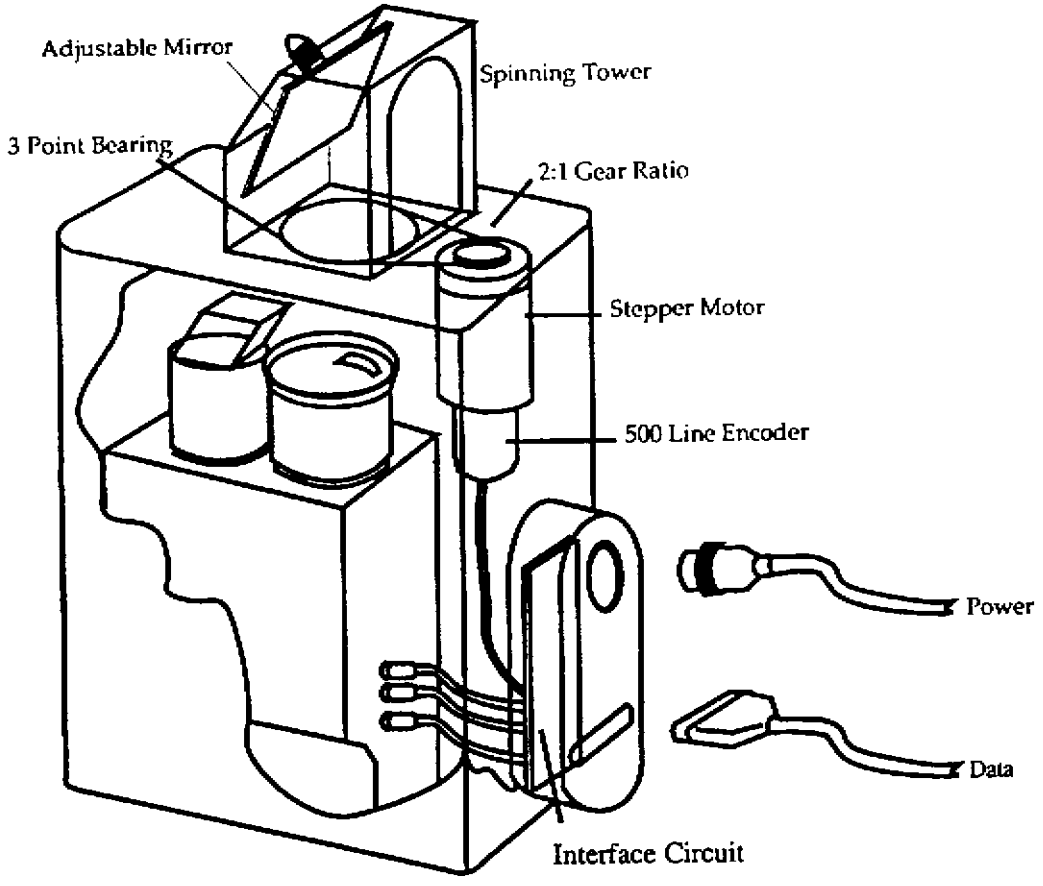

Figure 68: Scanner enclosure

The noteworthy components are as follows:

- Rotating Tower: A rotating tower at the top of the scanner enclosure, contains a mirror that reflects the transmitted and receiving beams. The tower is driven by a cog belt powered by a motor. There is a 2:1 gearing ratio between the motor and the tower. A four point bearing is used to hold the tower rigidly.
- Motor/Encoder: A DC stepper motor is used to drive the rotating tower. The encoder allows for the motion control to determine position of the motor shaft to within 0.2 degrees.
- Interface Circuit: This circuit is responsible for relaying power and data between the scanner and the remote PCE.
- Tower Motion Sensors: These sensors, mounted directly below the tower, have two functions. Firstly, they ensure that if the speed of the tower falls

below a minimum speed, the laser is disabled from firing, so as to not repeatedly fire the laser at one spot. This is a safety feature designed to minimize exposure to eyes, in case the tower stalls for some reason. A second sensor is used to mark an angle (between the two sensors) in which the laser can be selectively disabled. Typically, the laser scanner is mounted on the front of a vehicle, and the field of interest is in the 180 degree area in front of the vehicle. The vehicle itself will block the rear portion of the 360 degree scan area. The second tower motion sensor enables the determination of whether the tower is pointing out in front of the vehicle or towards it. This information can be used to disable the rangefinder from firing when the tower is pointing towards the vehicle. The second motion sensor is mounted 170 degrees away from the first one. This is used to indicate a "deadband" on part of the 360degree view (Figure 69).

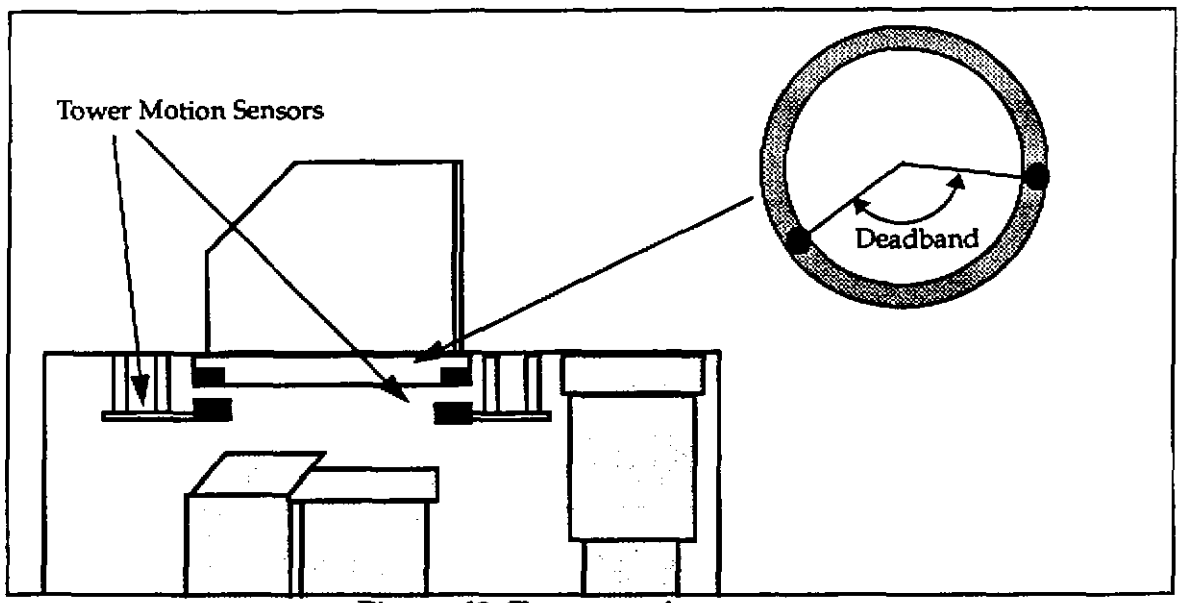

Figure 69: Tower motion sensors

8.1.2.3. Power and Communication Enclosure

As mentioned previously, the power and control signals are not generated in the scanner enclosure. Instead a separate enclosure has been designed (PCE). When the cyclone is mounted on the front of an autonomous vehicle, the PCE is rack mounted in the interior in a 19" rack. The PCE houses the following:

- Data Buffer Circuit: generates the control signals for the rangefinder, and buffers the range data till a complete scan can be transmitted to the host computer.
- Motor Amplifier: generates the power and control for the motor that drives the tower.
- Power Supply: Converts the AC voltage to DC for use by the buffer circuit (12 V @0.25 A, 5V @3A) and to power the rangefinder (12 V@ 1A) and the

interface circuit (5V @ 0.1 A).

The configuration of the PCE is shown in Figure 70.

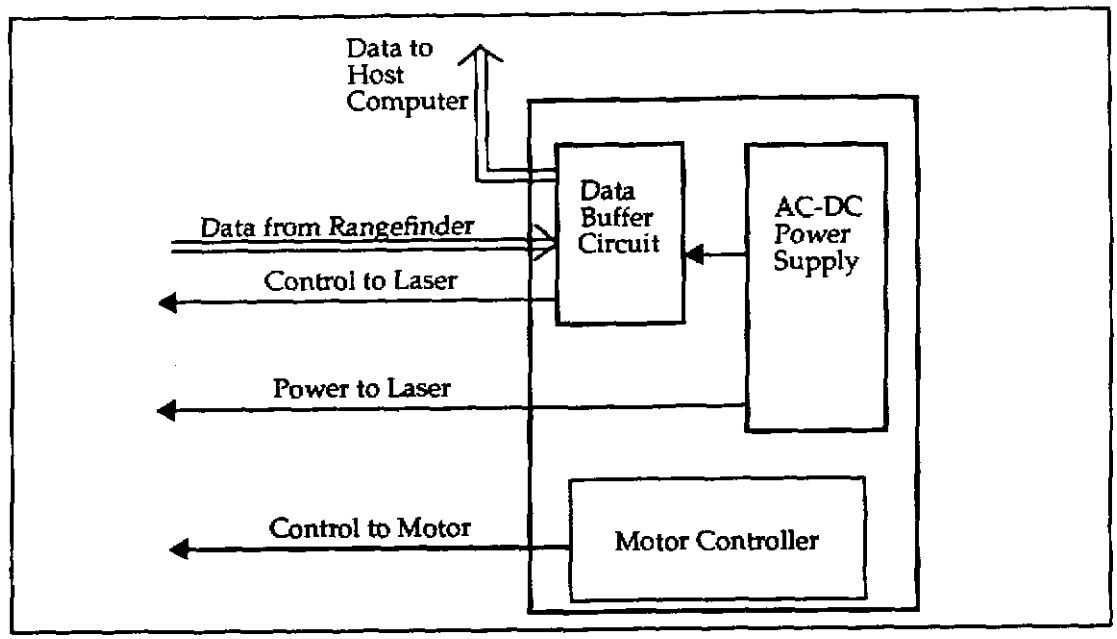

Figure 70: Power and Communication Enclosure

8.1.2.4. Motion Control

It is necessary to precisely control the speed of the scanner tower and hence the speed of the DC motor used to drive the tower. We were looking for a motion control system that afforded us the following:

- Precision of control: max 0.1% error in rotational velocity
- Simplicity: a simple stand-alone system that can communicate over a serial interface for high level commands and diagnostics.
- Ability to synchronize: each data set should be distinct, i.e, it is necessary for the motor to be repeatedly synchronized with the zero angle.

We chose a high resolution DC stepped motor[1] that is controlled by a single stand-alone unit providing all the necessary control and power to drive the motor. This device has 25000 steps/rotation and is comparable in smoothness to the DC servo motor. This motor provides 45 in/oz. of torque. Its top speed is 1800 RPM corresponding to 15 rotations/sec of the tower. Specifications of commands like motor speed and motor diagnostics are achieved through a serial interface on the motor controller.

An encoder is mounted on the shaft of the motor and provides three separate lines from which the position of the motor shaft can be precisely determined. Channels "A" and "B" produce a pulse train of 500 pulse/rotation that is 90 degrees out of phase. This provides 2000 edges (1000 rising and 1000 falling) from which the angular position of the motor can

1. Manufactured by Compumotor Inc.

be determined. The 'Z" channel produces a pulse whenever the motor completes one complete rotation.

8.1.2.5. Electronic Design

There are two main circuits: the *Interface Circuit* that resides in the scanner enclosure, and the *Data Buffer Circuit,* that resides in the PCE. The interface circuit transmits range data as it is obtained to the data buffer circuit via differential drivers and receivers. Range data are collected by the Data Buffer Circuit until the tower makes one full revolution. These data comprise one scan. When a scan is complete, the buffer circuit signals the host computer, whereupon the entire scan is transferred to the host computer for processing.

Interface Circuit

The Interface Circuit has three functions. First, it acts as a safety monitor. It uses one of the tower motion sensors to sense if the tower has stopped rotating. In this case, the LASER_INHIBIT signal is pulled high and the rangefinder is disabled from firing. The second function is to disable the laser from firing for part of the 360 degree scan area as mentioned earlier. This feature extends the life of the laser diode and also makes it possible to rotate the tower at a faster rate because the 7200 Hz firing rate is a RMS specification. However, for applications where a 360 degree scan is needed, the disable feature can be turned off.

The third function of the circuit is to convert signals between single ended and differential form. TTL signals (range measurements) from the rangefinder are differentially transmitted to the buffer circuitry, and differentially transmitted signals from the buffer circuitry are converted to TTL levels. That is, each signal is transmitted on two separate lines, one of which is a reference ground. Ambient noise affects both lines equally, but difference between the two lines is preserved. This procedure prevents noise contamination along the cable connecting the scanner and the PCE.

Data Buffer Circuit

The Data Buffer Circuit has three functions: to synchronize firing of the laser with the angular position of the tower, to buffer range data from one complete scan, and to transmit the scan to a host computer. Scan data are collected and stored in two memory banks. This avoids shared memory and synchronization problems between scan storage and scan transmission. Data for a new scan is stored in one bank while the previous scan is being transmitted from the other bank. This allows for more effective utilization of the CPU on the host computer since the host computer receives a whole scan at a time, rather than having to fetch one range measurement at a time.

Synchronization of the scans, i.e. correct detection of the start of a new scan is accomplished by the use of two quadrature pulse trains issued by the encoder: the Z and A channels. The Z channel is asserted once per motor shaft revolution. The A channel pulses 1000 times per revolution of the motor shaft. One additional signal is needed to fully synchronize the scan field with the encoder signals. Since there is a 2:1 gearing ratio between the motor and the tower, two pulses are received on the Z channel and 2000 pulses are received on the A channel per rotation of the tower. It is therefore not possible

to differentiate between the beginning of the first half of the scan and the beginning of the second half. To fully synchronize the scan field, the DB (dead band) signal generated by the interface circuit is used. The DB signal, used to disable the laser from firing in the back half of the scan, allows the differentiation of the front and back halves of the scan. The Z and DB signals together indicate the beginning of the scan.

The second task, buffering data for one complete scan, is accomplished through the A channel. The 2000 pulses of the channel are divided by either 2, 4, 8, or 16 (selected through DIP switches on the circuit board). This allows the number of data points per scan to be varied between 1000, 500, 250, and 125. The divided signal is used to trigger the laser rangefinder at appropriate angular intervals, and to store the resulting range data in memory.

The final task of the buffer circuit is to transmit the scan data to a computer for processing. Completed scans are signaled by the Z and DB signals (the beginning of a scan is also the end of a previous one). Upon sensing a completed scan, an interrupt request line is asserted and remains so asserted until either the mirror has made half a revolution or the computer acknowledges the interrupt. In the first case, the half revolution of the mirror is signaled by a subsequent Z pulse and indicates a time out condition; the computer has failed to respond and the data are lost. Normally, the interrupt is acknowledged by the host computer and upon its receipt, the data are put on the bus for the host to read.

8.1.3. Cyclone- Discussion

Figure 71 shows data from a single scan taken from the Cyclone while it was mounted on a mobile robot navigating through an underground mine intersection. This scan consists

of 1000 range measurements.

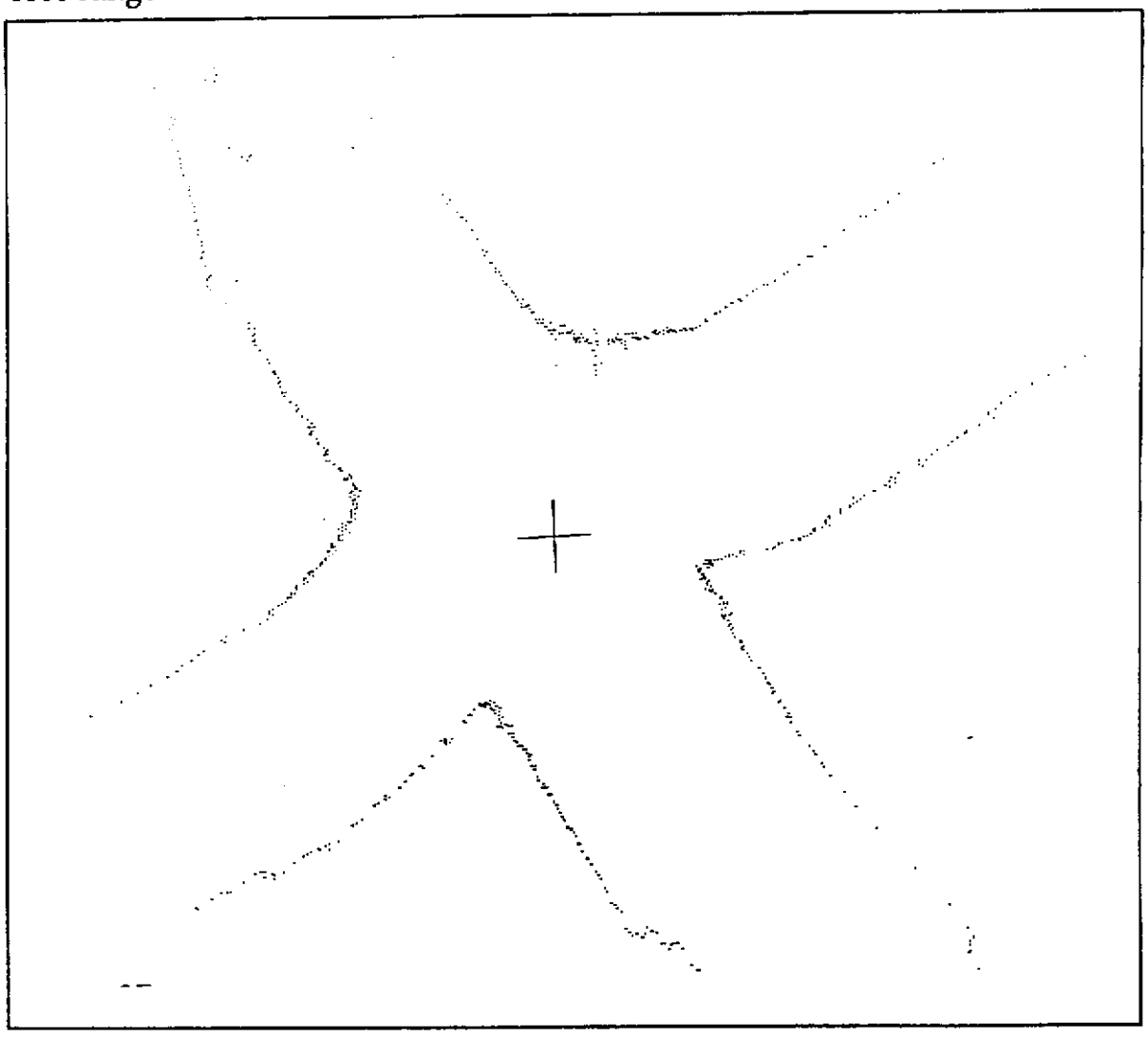

Figure 71: Sample data obtained from the Cyclone

Development of a stable system was complicated by several factors. Firstly, we were plagued by electrical noise problems. The stepper motor used to drive the scanner tower was the largest source of noise. Even after much shielding, data communication was constantly contaminated. This problem was solved by differential transmitting all signals between the scanner enclosure and the PCE.

We had a hard time finding a glass dust cover for the front of the tower. We found that quite often laser pulses were reflected back from the glass cover and were accepted as legal echoes. This was despite many efforts to treat the window with special coatings matched to the frequency of the laser, and to tilt the window. This problem has never been solved and it was determined that a modification in the rangefinder itself is necessary to compensate for this phenomena. For our experiments, we ended up manually removing a metal dust cover as necessary. Under most operating conditions (i.e. in the absence of a lot of dust), motion of the tower is sufficient to keep dust from entering the scanner.

A different set of problems are application dependant. For example, at high speeds, the lines traced by a scan on the ground (while mounted in the downward looking configuration) are not orthogonal to vehicle motion but lie at a velocity dependant angle to it. This is due to the relatively large amount of time taken by the rotation of the tower; the tower speed must be increased proportional to the higher vehicle speeds to prevent this effect. Another problem is due to the pitching motion of the vehicle that the Cyclone is mounted on. Given the small vertical field of view, the scan could completely miss the road (and hence any obstacle) or give the erroneous impression that the road surface lies much closer than expected. It is necessary to actively servo the pitch axis of the scanner to maintain a desired downward looking angle. This issue was also never dealt with.

Cyclone has proven to be a much larger success for applications other than the ones it was designed. Recent results in map based position estimation when mounted on a slow moving vehicle, navigating in underground mines, are unprecedented[26]. The range and the accuracy afforded by the system make it ideally suited for this sort of application.

8.2. NavArm

The NavArm is a large rotating arm that was designed to simulate vehicle motion in obstacle detection experiments. An arm of adjustable length rotates around a central bearing. A platform is located at the extremity of the arm and is mainly designed to carry sensors. The arm can make an indefinite number of rotations since any sensor data from the sensors on the rotating platform are routed through a slip-ring.

The top speed of this device is 11 m/s with an acceleration of 0.3 g with a 70 kg. sensor load on the platform. The arm is driven by a DC servo motor (3000 rpm at maximum) coupled with an orbital drive that has a 140:1 gear ratio and negligible backlash. A 200 line encoder is mounted on the motor which means that the arm can be positioned to within a centimeter.

We used NavArm to test the collision avoidance programs as they were being ported to a real-time environment. An object (1m X 1m) was rotated in and out of the collision zone set up by the collision avoidance programs. We were able to rotate the NavArm with several different configurations of collision zones and speeds and were able to success-

fully test the 2D obstacle detection scheme.

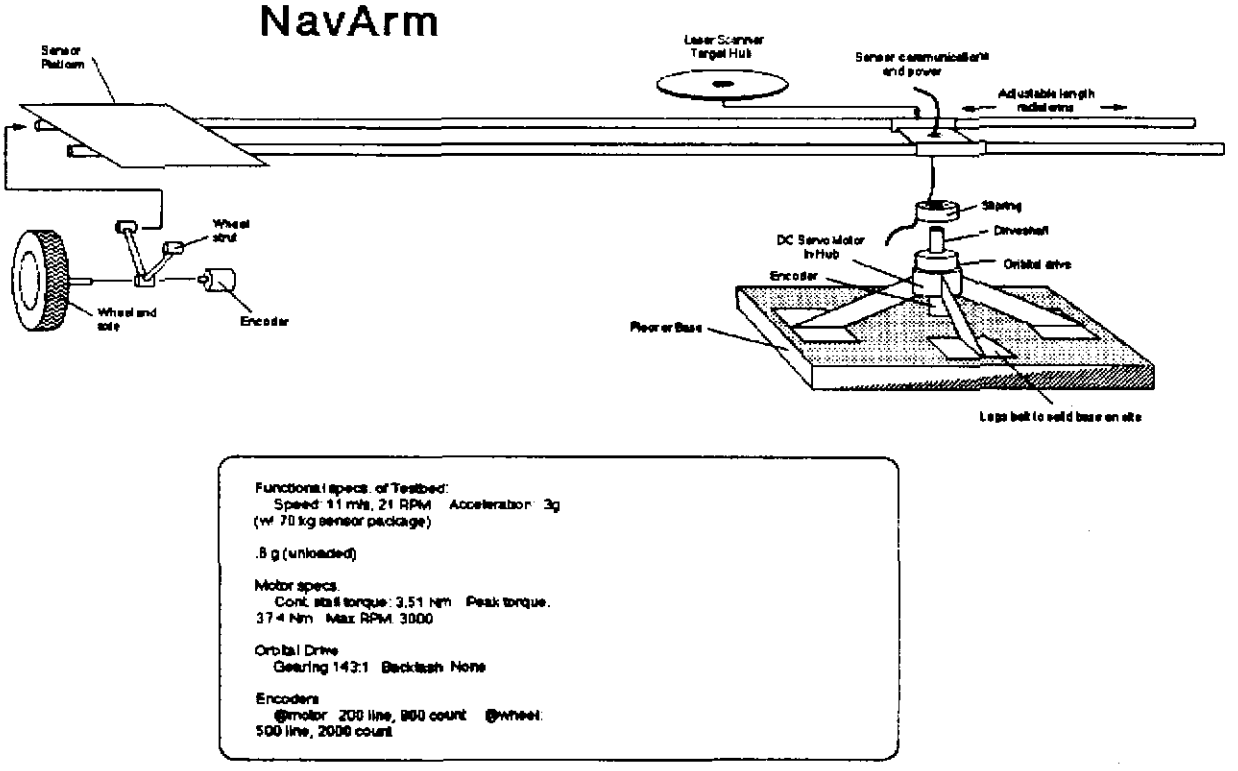

Figure 72: NavArm- a test device for obstacle detection

8.3. Vehicle Positioning System

Both the path tracking and obstacle detection applications require explicit vehicle position. This was accomplished by a combined inertial and satellite positioning system that was custom made commercially for our application. The device consists of the three main modules:

- Inertial guidance: three ring laser gyros were used along with a odometer on the drive shaft to produce vehicle position. Although the device is of high precision (1% of distance travelled), position estimates tend to degrade with time.
- Satellite positioning: a GPS receiver is used to sense the position of Navstar satellites and four sightings are used to determine vehicle position. In contrast to inertial guidance, satellite positioning is not as accurate, but does not worsen with time.
- Integration Module: this module takes position estimates from inertial guidance and satellite positioning and filters them. The result is transmitted on a high rate serial line to the host computer.

The device delivers position, heading, roll, pitch, yaw and a cumulative measure of distance travelled, every 50 msecs.

9. Summary

In this section we discuss final results obtained and briefly summarize some of the lessons learned over the course of the project.

9.1. Program Results

We have demonstrated path tracking at speeds up to 35 km per hour and a minimum radius of curvature of 200m. The vehicle tracked other paths with smaller radii of curvature (40 m) at 4 m/s. Obstacle Detection using the Cyclone scanner was demonstrated at 4m/s. We were able to reliably detect the same obstacles in simulation (with a 256 line range scanner) at speeds up to 10 m/s. We have also demonstrated obstacle avoidance at 1.5m/s in experimentation with obstacles 0.5 m wide and 1 m tall on a level 20 m wider roadway. In simulation we have been able to increase the speed by a factor of 3.

A scanning laser scanner was developed to serve as the primary sensor for obstacle detection and obstacle avoidance. We have also developed utilities for range data simulation.

9.2. Some Lessons Learned

There several valuable lessons that we have learned from our experiences over the course of the project. Perhaps the most important is that simple, superficial models work reasonably well in simulation and under low performance requirements. Higher performance specifications force the issue. We have found that unless the problem is not represented in sufficient depth, the resulting solutions are brittle– they only apply to a very narrow range of operating conditions. This distinction has surfaced in both obstacle detection and path tracking. In obstacle detection, we found that correlation is not sufficient to safeguard vehicle operation. It was not sufficient to detect if something had changed in the vehicle's environment, rather the road had be modeled explicitly with parameters for grade, bank and crown to determine whether obstacles existed. We also found that kinematic models are not sufficient for tracking. Geometric methods like the "quintic" method, ignore considerations such as mass, friction and cornering stiffness. Experimental results have shown that without explicit modeling of these parameters, only mediocre performance can be obtained.

A single, good development environment can save a tremendous amount of time. Ideally, the development environment should offer a variety of tools like graphics primitives, multiple windows and source level debuggers. Most importantly, the development environment should be very similar, if not identical to, the target environment

With many people working on various parts of a system, interfaces should be defined rigorously. By defining clear-cut interfaces and conventions, programmers have a distinct set of criteria by which their programs can be evaluated. While this sounds like just good software engineering, its importance cannot be stressed enough.

Three other issues are worth mentioning in regard to the architecture developed: generality, robustness and flexibility.

Generality: It is desirable to have an architecture that can be used for a variety of robotic

tasks. The FastNav architecture, however, is only general to the extent of specifying loosely coupled interaction among separate 'functions'. Even though the lowest level ensures message passing, memory arbitration, synchronization, and scheduling for a variety of systems, each such system needs a complete redesign based on the interaction necessary between functions. The entire task to be performed must be broken down into a deterministic sequence of actions, and it is from this sequence that the substance of the design evolves. Handpicking priorities and processor allocation is tantamount to hard-wiring an operating system. Such a design methodology is dissatisfying because deterministic task specification is a fundamental part of the design process. One would hope for an architecture that would accept more general task specifications. A major improvement to our system would be to incorporate a scheme to reason about the priorities that various modules of the system are assigned based on the current task that the robot is engaged in.

Robustness: Given Murphy's Law, any system that relies on a singular method or sensor for its performance will fail. At the other extreme it is possible to add so many features to a system that the resulting system itself is not very robust. Our architecture has marginally addressed this issue in two ways:

- The most conservative plan of action is chosen. Several tasks contribute to the determination of vehicle control parameters but only the most conservative of these inputs is chosen
- The default health of critical modules is assumed to be dysfunctional. Hence, these modules are monitored specifically to ensure that they are alive without waiting for exception conditions to be signaled.

Flexibility: We are faced with a common trade off between performance and flexibility/generality. Traditional AI approaches assume that if there is a discrepancy between the model and the observed world, then it is only necessary to apply the appropriate operator to correct the problem. This sort of approach while applicable widely, can have a serious cost in terms of performance. Alternately, the decision criteria could be built into the control system at any level of abstraction, gaining performance but at the cost of flexibility. We have had to visit the performance versus flexibility issue in the design of the FastNav architecture. There are two philosophies-- one suggests that an architecture should be designed to be general even at the cost of sacrificing performance, while the opposing view is that an architecture should be geared towards a particular task without much consideration to how the solution generalizes. Both approaches appear to be valid depending on the commitment of the research. In the development of simple automatons, the latter approach seems to be justified especially if it provides high utility. Clearly, however, given the overall commitment to building complete machines that can work beyond the bounds of a deterministic world, neither is sufficient on its own, and a compromise is necessary.

Acknowledgments

Acknowledgment is due to the entire FastNav team who put in an extraordinary effort. Dong Hun Shin was responsible for the development of the path tracking algorithms. Richard Clow developed the early schemes for obstacle detection. Dai Feng implemented the obstacle avoidance algorithms on the NavLab. Jay West was responsible for system software on the integrated system and also did much of the electronic design for the laser scanner. Wen Fan Shi implemented low level primitives and I/O on the real-time systems. Gary Shaffer was responsible for imaging utilities, range simulation, and implementation of the primitives under simulation. Bao Xin Wu implemented frequently used algorithms on an array processor. Paul Keller implemented the alternate obstacle detection scheme. Lonnie Devier was a tremendous help in much of the experimentation work that we did. Sanjiv Singh led the research, and implemented the primary obstacle detection scheme.

In addition the authors would like to thank Gary Baun and Jim martin for their help in fabrication of test devices. Bryon Smith helped with several electronic design projects. Thanks to Roy Taylor for help in making this document more readable.

Last but not least, acknowledgment is due to Red Whittaker for his vision and confidence in an inexperienced group, without which this work would not have been possible.

References

1 Ballard, D.H. and Brown, C., *Computer Vision*, Prentice-Hall Inc., 1982.

2 P. Besl, "Range Imaging Sensors," Technical Report, General Motors Research Lab. GMR 6090, March 1987.

3 J. Borenstein, Y. Koren, "Vector Field Histogram- Fast Obstacle Avoidance for Mobile Robots," IEEE Journal of Robotics and Automation, Vol. 7, June, 1991

4. M. J. Daily, J. G. Harris, K. Reiser, "Detecting Obstacles in Range Imagery," In *Proceedings DARPA Image Understanding Workshop*, February, 1987.

5 J. C. Dixon, "Linear and Non-linear Steady State Vehicle Handling," in *Proceedings Instn. Mech Engineers*, Vol. 202, No. D3, pp. 173-186, 1988.

6 E. Donges, "A Two-level Model of Driver Steering Behavior," Human Factors, 20(6), pp. 691-707.

7 H. Dugoff, P. S. Fancher, and L. Segel, "An Analysis of Tire Traction Properties and Their Influence on Vehicle Dynamic Performance," *SAE paper 700377*, 1970.

8 R. O. Duda and D. Nitzan, "Low Level Processing of Registered Intensity and Range Data," in *Proceedings, 3rd International Joint Conference on Artificial Intelligence*, 1976.

9 R. T. Dunlay, "Obstacle Avoidance Perception Processing for the Autonomous Land Vehicle," *Proceedings IEEE ICRA*, Philadelphia, 1988.

10 K. Dowling, R. Guzikowski, H. Pangels, S. Singh, W. Whittaker, "NavLab: An Autonomous Vehicle," Technical Report, CMU-RI-TR-87-24, Robotics Institute, Carnegie Mellon University, 1987.

11 D. Feng, S. Singh, B. H. Krogh, "Implementation of Dynamic Obstacle Avoidance on the CMU NavLab," in *Proceedings IEEE Conference on Systems Engineering*, Pittsburgh, PA, 1990.

12 D. Feng, "Satisficing Feedback Strategies for Local Navigation of Autonomous Mobile Robots," Ph.D. Thesis, CMU, 1989.

13 D. Feng and B. H. Krogh, "Dynamic Steering Control of Autonomous Mobile Robots," to appear in Journal of Robotic Systems, August, 1991.

14 M. Hebert, T. Kanade, "3D Vision for Outdoor Navigation by and Autonomous Vehicle," In Proceedings of Image Understanding Workshops, Morgan Kaufman, Cambridge, Mass, 1988.

15 M. Hebert, "Building and Navigating maps of Road Scenes Using an Active Sensor," *in Proceedings IEEE International Conference on Robotics & Automation*, 1989.

16 R. A. Jarvis, "A Perspective on Range Finding Techniques for Computer Vision," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 5, March 1983.

17 R. A. Jarvis, "A Laser time-of-flight range scanner for robotic vision," Technical Report Australian National University, TR-CS-81-10, 1981

18 R. A Lewis and A. R. Johnston," A Scanning Laser Rangefinder for a Robotic Vehicle", in *Proceedings 5th International Joint Conference Artificial Intelligence*, 1978.

19 Moravec, H., "Sensor Fusion in Certainty Grids for Mobile Robots," AI Magazine, Summer 1988.

20 Nelson, W. L., "Continuous Steering Function Control of Robot Cart," IEEE Transactions on Industrial Electronics, Submitted for publication 1988, AT&T Bell Laboratories.

21 U. Regensburger, V. Graefe, "Object Classification for Obstacle Avoidance," in *Proceedings SPIE Symposium on Advances in Intelligent Systems*, 1990.

22 H. Sakai, "Theoretical and Experimental Studies on the Dynamic properties of Tyres," Part 1: Review of Theories of Rubber Friction, *International Journal of Vehicle Design*, Vol. 2, No. 1, pp78-110, 1981.

23 S. Sedas and J. Gonzalez, "Analytic and Experimental Characterization of a Radial Laser Range Finder," Technical Report to appear, Robotics Institute, Carnegie Mellon University.

24 S. Sedas and J. Gonzalez, "Improvement in Robot Position Estimation Through Accurate Sensor Characterization," Submitted to IFAC Symposium on Intelligent Components and Instruments for Control Applications, 1992

25 G. Shaffer, "WB3: A 3D Range Scanner Simulator," Internal FRC Report, Carnegie-Mellon University, August, 1989.

26 G. Shaffer et al, "Position Estimator for Underground Mine Equipment," in *Proceedings 10th WVU International Mining Electrotechnology Conference*, Morgantown, W. Va., July 1990.

27 Sheridan, T. B., "Three Models of Preview Control," IEEE Transactions on Human Factors in Electronics, Vol. HFE-7, No. 2, pp91-102, June, 1966.

28 D. H. Shin, S. Singh, "Vehicle and Path Models for Autonomous Navigation," in *Vision and Navigation: The Carnegie Mellon NavLab*, Editor Chuck Thorpe, Kluwer Press, 1990.

29 S. Singh, P. Keller, "Obstacle Detection for high Speed Autonomous Navigation," in *Proceedings IEEE International Conference on Robotics and Automation*, Sacramento CA, April 1991.

30 S. Singh, J. West, "Cyclone: A Laser Scanner for Autonomous Vehicle Navigation," Technical Report, Robotics Institute, Carnegie Mellon University, CMU-RI-TR-91-18, August 1991.

31 M. Tomizuka, "The Optimal Finite Preview Problem and Its Application to Man-Machine Systems," Ph. D. Dissertation, Department of Mechanical Engineering, MIT, Cambridge, Mass., Sept., 1973.