Integrated Mobile Robot Control

Omead Amidi

CMU-RI-TR-90-17

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

May 1990

This research is sponsored in part by contracts from DARPA (titled "Perception for Outdoor Navigation" and "Development of an Integrated ALV System"), by NASA under contract NAGW-1175, by the National Science Foundation contract DCR-8604199, and by the Digital Equipment Corporation External Research Program.

Contents

1 Introduction **1**
1.1 Improving and building upon previous work 1
1.2 Navlab Background 2
1.3 Architectural Overview 3

2 State Maintenance System **5**
2.1 Coordinate Frame 5
2.2 State Maintenance System Operation 6
2.3 Dead Reckoning 6
2.4 Using other sensors 9

3 The Clients **11**
3.1 Path Specification 11
3.2 The Mapper 12
3.3 The Tracker 13
3.3.1 Control Theory Approach 13
3.3.2 Quintic Polynomial Fit 16
3.3.3 Pure Pursuit 21
3.4 Velocity Regulator 22
3.5 Software Interface Server 23
3.6 Hardware Interface Server 24

4 Actuation System, Watch Dog, and Request Maintenance System **26**
4.1 Actuation System 26
4.1.1 Steering Actuation 26
4.1.2 Drive Actuation 26
4.2 The Watch Dog 27
4.3 Request Maintenance System 28

5 Conclusions **29**
5.1 Acknowledgements 29

A Hardware Details **30**
A.1 Main processor 30
A.2 Communications 30
A.3 Low-level PID controller 30

List of Figures

1.1 The CMU Navlab Autonomous Navigation Testbed 2
1.2 Architectural Overview 4

2.1 Navlab's coordinate frame 5
2.2 Structure of "State Maintenance System" 6
2.3 Dead Reckoning geometry 7
2.4 Kinematic Model of the Vehicle 7
2.5 Plot of Steering angle vs. Encoder position 8
2.6 Dead Reckoning performance with circular paths (speed: 3.5 m/s) 8
2.7 Dead Reckoning performance with non-circular paths (speed: 3.5 m/s) 9

3.1 Clients of the Navlab Controller 11
3.2 Effect of instantaneous change in curvature 12
3.3 A list of path points supplied to the mapper 12
3.4 Compensation for error in X 14
3.5 Compensation for error in Heading 14
3.6 Incremental goal points of the Control Theory tracker 14
3.7 Control Theory tracker performance at 2.5 m/s 16
3.8 Tracking with quintics at 3.0 m/s (simulation) 20
3.9 A sequence of Pure Pursuit cycles 21
3.10 Fitting and arc for Pure Pursuit 21
3.11 Pure Pursuit tracker performance at 3.5 m/s 25
3.12 Effect of reducing the requested Velocity 25

4.1 Block diagram of the Actuation System 27

Abstract

This report describes the structure, implementation, and operation of a real-time mobile robot controller which integrates capabilities such as: position estimation, path specification and tracking, human interfaces, fast communication, and multiple client support. The benefits of such high-level capabilities in a low-level controller was shown by its implementation for the Navlab autonomous vehicle. In addition, performance results from positioning and tracking systems are reported and analyzed.

Section 1

Introduction

Real-time mobile robot controllers have usually been designed with an emphasis on control theory ignoring the importance of system integration. This report demonstrates that useful mobile robots require a real-time controller with a wide range of capabilities in addition to control theory. These capabilities include: position estimation, mapping and tracking of paths, human interfaces, fast communication, multiple client support, and monitoring vehicle status for safety and debugging. An architectural framework supporting these capabilities has been designed and implemented. Using this framework, individual modules such as a position estimator, a path tracker, a mapper, network servers and other crucial elements have been successfully integrated into a controller for the Navlab autonomous vehicle. Issues regarding the use of an Inertial Navigation System for position estimation are discussed and the positioning data is compared to dead reckoning techniques. Three path tracking strategies: pure pursuit, quintic polynomial fit, and control theory approach are analyzed and compared. In addition the operation of a smooth velocity profiler for the Navlab is discussed. The context of the research, the architecture, its implementation, and other performance results from experiments are discussed.

1.1 Improving and building upon previous work

Other researchers have produced important parts of robot controllers:

- Dickmanns and Graefe [3] have built an elegant control formulation for driving an autonomous van on well-structured highways. Their approach has been to build a special purpose controller with real-time perception at its heart. They have achieved impressive performance at speeds up to 100 kph. Their work is carefully tuned to the real-time demands of a single dedicated perception module.

- Feng [5] formulated and solved a local navigation problem, incorporating the vehicle's kinematic and dynamic constraints and environmental constraints such as obstacles, and proposed the conversion of all these constraints into control constraints to allow real-time performance. This approach drove the Navlab around obstacles at speeds up to 1.5 m/s.

- Guzikowski [6] presented an approach to control computing for autonomous mobile systems, and implemented the first Navlab controller. His approach stressed hardware independence and prototyping. The controller was basically a front-end to the actuators and used shaft encoders to reckon a vehicle position which was only used internally by the controller.

- Shin [12] and Singh built a path tracker which used an Inertial Navigation System (INS) for positioning and the old Navlab controller as the main interface to the actuation system. This system was not integrated with the rest of the systems on board the Navlab and was essentially built for high-speed dedicated tracking performance. Some tracking strategies used in this report were inspired by their work.

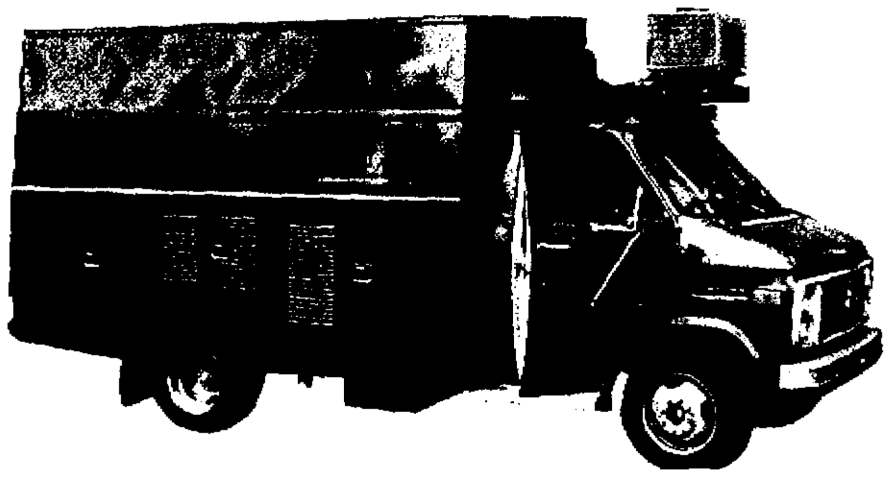

Figure 1.1: The CMU Navlab Autonomous Navigation Testbed

Each of these researchers made a strong contribution to a specific part of mobile robot control. To build the controller described in this report, a framework was developed that incorporated previous work on key components. Integrating these components produced a powerful research platform for further analysis of the system capabilities. The components were tested, tuned, and modified to match the capabilities and requirements of the Navlab's navigation systems. The resulting controller provides high-level abstractions of low-level operations, creating simple yet powerful interfaces to the high-level systems.

1.2 Navlab Background

From the outside, the Navlab (short for Navigation Laboratory) looks like a standard commercial van, with a rooftop air conditioner, plus one or more video cameras and a laser rangefinder mounted over the cab. On the inside, it looks more like a machine room, with five electronics racks, 20kw of on board power, and miscellaneous consoles and monitors. Over time, the Navlab has carried Sun 3's, Sun 4's, several generations of the CMU / GE Warp supercomputer, various specialized real-time controllers, gyrocompasses, an inertial navigation system, and a satellite positioning system. It has a hydraulic drive system and it can be propelled or stopped by opening or closing the flow of hydraulic fluid through the transmission. Its steering wheel can be actuated by a servo motor mounted on the steering column.

The most important payload of the Navlab is the researchers. There is always a safety driver in the driver's seat, watching over the Navlab's autonomous runs. In the back, there is room for five researchers plus observers. The Navlab is a laboratory on wheels.

The Navlab is driven by several different road following systems. Current road following systems are:

- The Autonomous Mail Vehicle, or AMV system draws its inspiration from postal deliveries in suburban or rural areas, which follow the same route day after day. This type of system is an example of a broader class of applications which focus on map building and reuse, positioning, road following, and object recognition. The AMV project is investigating those issues, including strategies for using different sensors and different image understanding operators for the perception components.
- The Generic Cross-Country Vehicle (GX-CV) packages 3-D perception and local trajectory planning, into a solid foundation for off-road navigation. The basic GX-CV system enables the Navlab to travel in a general direction, planning and executing vehicle trajectories around obstacles and across rough terrain. Hooks in the GX-CV planner will enable missions such as cross-country traverse with global navigation, or cross-country mapping.

- SCARF, which stands for Supervised Classification Applied to Road Following, tracks roads by adaptive color classification [2]. SCARF runs in a loop of: classify image pixels, find the road model that best matches the classified data, and update the color models for classification. The simple models of road color and geometry make very few assumptions about the road, and allowing SCARF to run robustly even when following unstructured roads.
- The YARF system (Yet Another Road Follower) explicitly models as many aspects of road following as possible, for driving on structured roads [7]. Highways, freeways, rural roads, even suburban streets have strong constraints. Modeling these explicitly makes reasoning easier and more reliable. When a line tracker fails, for instance, an explicit model of road and shoulder colors adjacent to the line helps in deciding whether the line disappeared, became occluded, turned at an intersection, or entered a shadow. This kind of geometric and photometric reasoning is vital for building reliable and general road trackers.
- ALVINN, for Autonomous Land Vehicle in a Neural Net, is the newest of the road following programs, built by the CMU Connectionist group [11]. The weights in ALVINN's neural network hidden units are trained by driving the Navlab by hand. During the training phase, ALVINN inputs the camera image, and the steering angle from the human driver, at each time step. The image is pre-processed to enhance road contrast. The enhanced image and the steering angle are fed to a back-propagation algorithm that adjusts weights in the hidden units, until the weights settle to values that give the correct steering response for each input image.

1.3 Architectural Overview

An autonomous vehicle needs answers to the following questions before it can perform a useful mission.

- Where am I?
- Where do I want to go?
- How do I get there and not get hurt?

These questions were used as motivations for the architectural design of the vehicle controller. To answer the question "Where am I?" as accurately as possible, data from different sensors must be fused into a coherent set of vehicle status information. Recognizing the importance of this data, one major component of the control system is devoted to this task. The "State Maintenance System" interfaces to all sensors and continuously monitors and processes their data which it maintains in a buffer shared by other modules. This maintenance system is aware of the state of each sensor as well and attempts to sustain operation by changing modes in case certain sensors fail.

Other important questions such as "Where do I go?" and "How do I get there?" are answered by the "clients" of the controller. The clients are classified in two groups, the navigators and the observers. A navigator, is a module that among other things, uses the status information to make new decisions that will eventually cause vehicle motion. For example a map following algorithm is considered a navigator. The other group, the observers, consist of modules that only sample the status data without making any decisions that involves vehicle motion. For example, a map building program is a member of this group. Some clients are external to the controller. These modules need special interface servers to pose for them in the controller. For example, a vision system executing on a super-computer will have to communicate via the network to a software server in the controller to relay movement requests or receive positioning data. These relaying modules are server programs that can handle communications from multiple external clients. The order of servicing these clients is determined by the priority of the client set at connecting time. Therefore, at each cycle of operation, these servers act as a different client of the controller.

The final question "How do I get there and not get hurt?" is handled by the system "Watch Dog." The watch dog glances at the status information and interferes if it decides that the vehicle's safety is in jeopardy. For example if

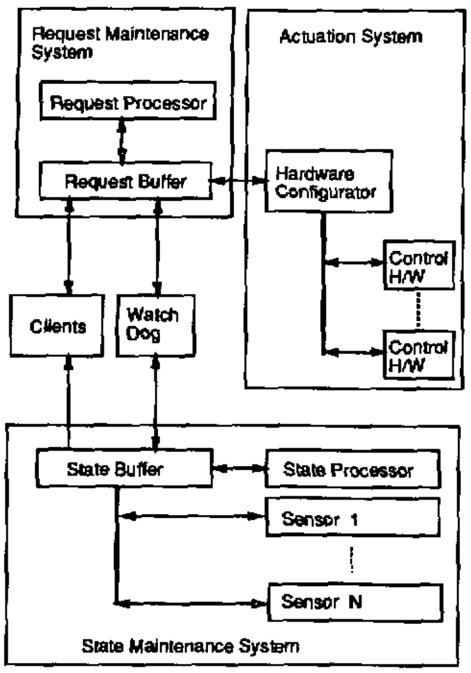

Figure 1.2: Architectural Overview

the vehicle pitch reaches a threshold value, the watch dog will intervene and override the client requests and stop the vehicle and report the reasons for its action. It is also responsible for generating messages or warnings regarding the vehicle's state which are valuable in system debugging and configuration.

All of this processing and data maintenance leads to a set of concrete actions to be performed by mechanical actuators of the vehicle to produce the proper motion. The "Actuation System" is the component of the system devoted to this task. It interfaces to external hardware that sends signals to actuators such as motors or solenoids. The Actuation System automatically adjusts the necessary parameters depending on the vehicle's status information. For example the gains of a PID controller must be updated with changing gears in the transmission.

The central bookkeeping for the clients, the watch dog, and the actuation system is done by the "Request Maintenance System" which consists of a large request buffer and a process that continuously keeps this buffer consistent.

Figure 1.2 displays the resulting architectural overview of the system consisting of the described building blocks. Notice the positioning of the watch dog. It is in parallel to the clients enabling it to easily intervene in case a client's decisions are considered undesirable. Also, this structural layout encapsulates details such as actuator configuration or sensor integration in isolated modules simplifying the communication interface among different components. This also allows the architecture to be reused on different vehicles by modifying the components directly dependent on specific vehicle characteristics. A controller with this structure has been implemented on the Navlab. The following sections describe each portion of this controller.

Section 2

State Maintenance System

Primarily, the task of the State Maintenance System is answering the "Where am I?" question for the mobile robot. This system uses the data from an array of sensors to produce an accurate estimate of the vehicle state. To discuss this system's operation, it is necessary to define the positioning information and the coordinate frame of the vehicle.

2.1 Coordinate Frame

To simplify the computation of the radius of curvature, the frame of reference of the four-wheeled robot has been chosen to be the center of its rear axle with the Y-axis pointing out of the front, the X-axis pointing to the right, and the Z-axis pointing upwards. Roll (ψ), pitch (θ), and yaw (ϕ) are defined as follows:

- *roll:* The projection of the angle from the world's to the robot's Z-axis on the *robot's* X-Z plane.
- *pitch:* The projection of the angle from the world's to the robot's Z-axis on the *robot's* Y-Z plane.
- *yaw:* The projection of the angle from the world's to the robot's X-axis on the *world's* X-Y plane.

These angles are orthogonal and describe the vehicle's orientation with respect to the world coordinate frame. Yaw is measured with respect to North, and roll and pitch are measured with respect to the ground plane. All angles increase in the counter-clockwise direction, which implies that a left turn by the vehicle signifies a positive radius.

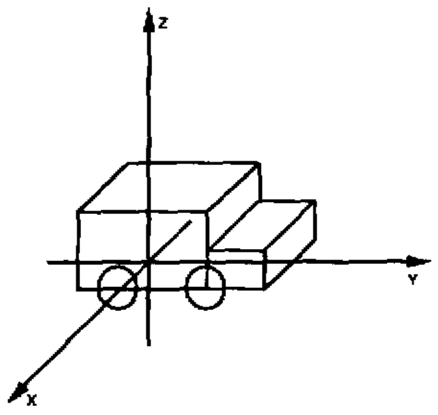

Figure 2.1: Navlab's coordinate frame

2.2 State Maintenance System Operation

The most valuable piece of information maintained by state maintenance system is vehicle positioning information. Referring to the needs of the perception and tracking systems, it was apparent that in addition to position and orientation data $(x, y, z, \psi, \theta, \phi)$ distance traveled (s), velocity $(\frac{ds}{dt})$, and radius of curvature $(\gamma = \frac{ds}{d\phi})$ of the robot are key information for navigation. The general structure of the state maintenance system is portrayed below.

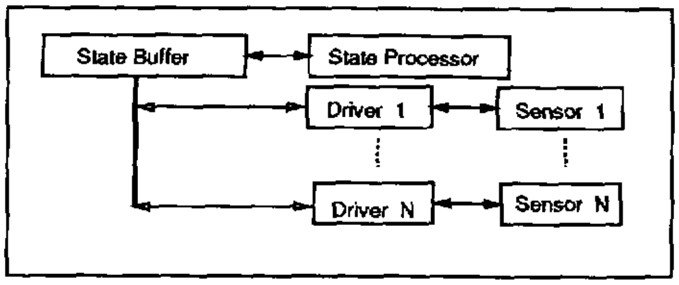

Figure 2.2: Structure of "State Maintenance System"

The state buffer is a centralized buffer that is continuously updated by the sensor drivers which interface to the available sensors. Each driver cycles at a frequency which is defined within the state buffer itself and is limited by the sensor's ability to deliver data. The drivers also mask the details of each sensor from the rest of the system and only deliver the relevant data to the state buffer. In addition to sensed data, the drivers update the state buffer with information about the sensor's state and operating conditions which affects the accuracy of the data from the sensor. For example, a driver is capable of detecting whether a sensor has over-heated and its data is erroneous.

The Navlab's positioning sensors are: An optical shaft encoder in the transmission reporting the absolute shaft position which is translated to distance traveled or velocity of the vehicle; An optical encoder on the steering column reporting the steering wheel position which is used to infer the vehicle's radius of curvature; Drive system sensors which report which gear in the transmission is engaged; An uncalibrated Inertial Navigation System (INS) with accurate, high precision heading, roll, and pitch data.

The state processor has the task of examining all the sensor data in the state buffer and merging it into one piece of data usable by the rest of the system. The following two sections describe the details of the state processor operation.

2.3 Dead Reckoning

Given that the exact curvature (γ) and traveled distance (s) of the vehicle are known, it is possible to calculate changes in x and y in world coordinates. Assuming that it is constant for the computation interval, the steering angle will provide the change in heading $(d\phi)$ from one point to the next. In other words, the assumption is that the vehicle moves from one point to the next along a circular arc. Therefore we have:

$$\begin{aligned} d\phi &= \gamma ds \\ r &= \frac{1}{\gamma} \end{aligned}$$

and in vehicle coordinates:

$$dx = -(r - r\cos(d\phi))$$

$$dy = r\sin(d\phi)$$

rotating back to world coordinates:

$$\begin{aligned} dx &= r\cos(\phi)[\cos(d\phi) - 1] - r\sin(d\phi)\sin(\phi) \\ dy &= r\sin(\phi)[\cos(d\phi) - 1] + r\sin(d\phi)\cos(\phi) \end{aligned}$$

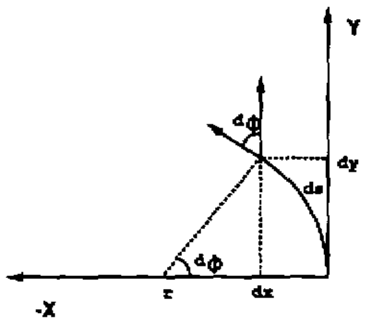

Figure 2.3: Dead Reckoning geometry

Provided that the computations are done in short intervals, the path between two points can be approximated as a straight line and:

$$\begin{aligned} dx &= -\sin(\phi)ds \\ dy &= \cos(\phi)ds \\ d\phi &= \gamma ds \end{aligned}$$

The frequency of this computation is proportional to the values of highest attainable curvature and highest speed for that curvature. As the sharpness of curves in the path increases, the length of the line segments must decrease to approximate the path more accurately.

The accuracy of this data is purely dependent on the errors in steering angle (α) and traveled distance estimation. Considering one wheel angle at the front, we have the following configuration:

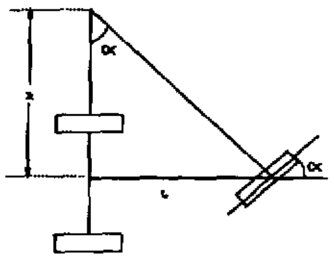

Figure 2.4: Kinematic Model of the Vehicle

which means that:

$$\alpha = \arctan(\frac{l}{r}) \tag{2.1}$$

The available data from the actuation system is the position of the steering wheel in encoder counts. Therefore, to estimate α, it is necessary to relate steering wheel position to α. To determine this relationship experimentally, the distance traveled was sampled from the wheel encoders and heading data was recorded from a gyroscope at short time intervals as the vehicle was driven on a wide variety of paths at different speeds. The ratio of ds to $d\phi$ is the sensed radius of curvature.

Provided that the interval's length is short enough, we can assume that the steering wheel position is essentially a constant during this interval. Therefore, we can associate the measured radius of curvature with the particular steering wheel position. The sensed curvature was stored in a table indexed by the encoder counts. For a better approximation of the steering wheel position, the encoder positions reported at the beginning and end of the time interval were averaged and this averaged number was used to index this table. Because of large number of encoder counts, it was sufficient to average the different measured radius values for the same encoder position. Figure 2.5 illustrates the data in this table. Overall the relationship appears linear with some uncertainty. That is, for the same sensed turning radius, there are multiple encoder positions. The linear regression line is plotted and the slope was taken as the linear constant relating encoder position and steering angle (α).

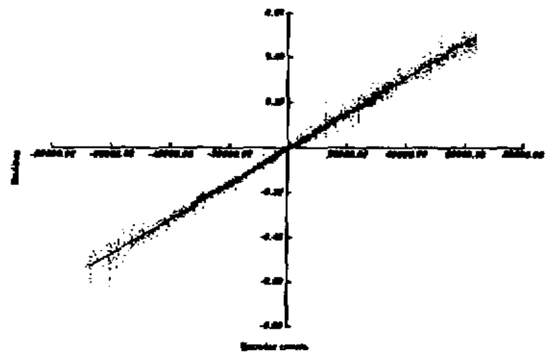

Figure 2.5: Plot of Steering angle vs. Encoder position

There are several possible reasons for the uncertainty in the estimated curvature from the steering wheel position. Tire pressure changes the circumference of the wheels making distance traveled data incorrect. Also, tire slippage and deformation have similar effects causing ***under/oversteering*** which varies the turning radius. Distortion in the sidewalls of the tires is considered to be a major contributor to these errors since it causes the tire threads to move in a different direction from the wheel rims. More detailed analysis of these factors and effects of vehicle dynamics are necessary to discover other sources for this uncertainty in the curvature estimation.

This dead reckoning of position works quite well for small distances and since the encoders are necessary for servo control, no additional hardware is necessary. The computations involved are simple which makes dead reckoning fast. Here are a few performance plots:

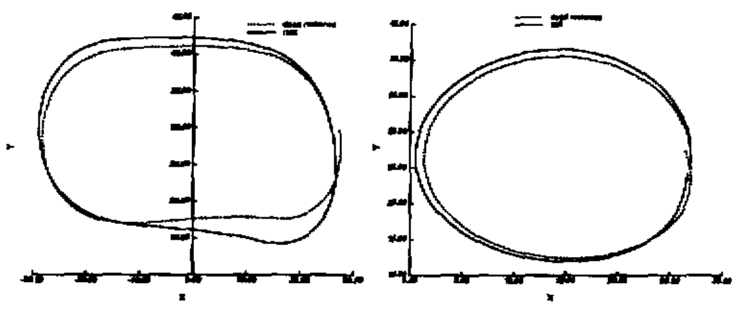

Figure 2.6: Dead Reckoning performance with circular paths (speed: 3.5 m/s)

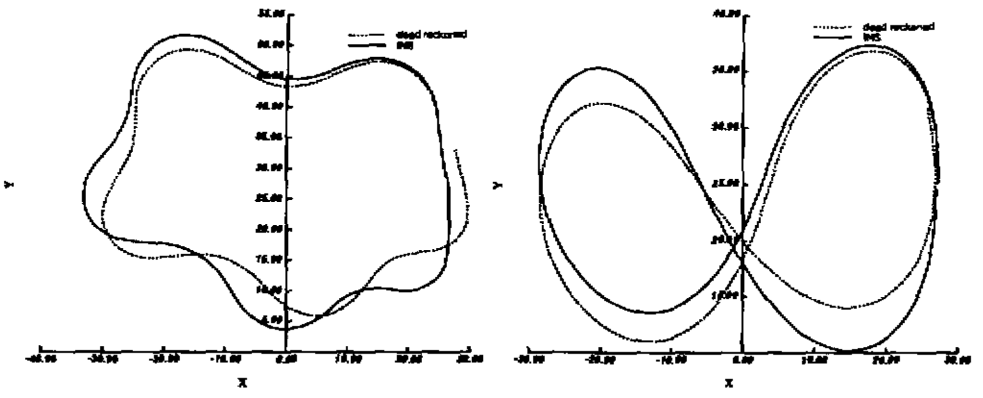

Figure 2.7: Dead Reckoning performance with non-circular paths (speed: 3.5 m/s)

In these experiments, the vehicle was driven on a closed path and dead reckoning data and INS data were recorded simultaneously. The solid lines represent the INS data which is an accurate representation of the vehicle's position. The dotted lines are the reported dead reckoned position. Note how the error accumulates as small errors in dϕ translate to big errors in position as they get multiplied by the radius. The magnitude of error increases as the steering wheel is moved back and forth.

A big deficiency of dead reckoning is the absence of elevation (z), roll (ψ), and pitch (θ) data. This makes compensation for the steepness of the road in the vehicle curvature estimation impossible.

2.4 Using other sensors

In order to eliminate the error accumulation from the dead reckoned position, an external data source is necessary. One approach is to use an Inertial Navigation System (INS) for determining the global position of the vehicle. Typically, these systems consist of a set of two or three gyroscopes and accelerometers. The knowledge about earth's movement and the latitude and longitude of the vehicle are used in addition to heavily filtered sensor data to compute a global position. This is done by double integrating the sensed accelerations from the accelerometers to reckon the (x, y, z) positions. Orientation data (ψ, θ, ϕ) are supplied by the gyroscopes. These systems require initialization with accurate initial global position for alignment. Therefore, an INS could be initialized using satellites via the Global Positioning System (GPS) or the initial position can be estimated using a map.

Determining position data for a slow moving vehicle with small accelerations requires extremely sensitive accelerometers. This translates to very noisy data which must be filtered to obtain the best estimated position. This is especially true for acceleration in the Z direction. In spite of all these considerations, although quite accurate, the precision of the positioning data is typically not better than one meter. On the other hand, dead reckoning is accurate for short distances. So, a good strategy is to use INS data for the low precision global positioning data and refine it by examining the shaft encoders for the local position between the two INS data points.

Navlab's INS was previously used on a howitzer barrel and although its orientation data (ψ, θ, ϕ) data is very accurate, its position data (x, y, z) is inaccurate due to calibration errors. Therefore, the state processor uses only the orientation data and distance traveled in order to compute a global position for the Navlab. This is done by looking up the initial position of the Navlab in a map and computing the small changes in (x, y, z) with a small change in distance (s). The basic assumption in these computations is that the vehicle's orientation remains constant for the small distance interval ds. Therefore, the recorded orientation data at the beginning of the small interval is used to calculate the x, y, and z coordinates at the end of the interval before updating the orientation data at the end point. The following equations calculate the changes in x, y, and z given the orientation data and ds.

$$dx = ds\cos(\theta)\sin(\phi)$$

$$
\begin{aligned}
dy &= ds\cos(\theta)\cos(\phi) \\
dz &= ds\sin(\theta)
\end{aligned}
$$

Since none of the INS's accelerometers are currently usable, the behavior of the suspension affects the accuracy of vehicle positioning. For example, a long continuous braking compresses the front springs making the shell tip forward and giving the false impression that the vehicle is going downhill. Although there are other factors to consider, as a low-level approximation, it is sensible to assume that the body deflection is proportional to the acceleration of the vehicle. One strategy to compensate for errors in pitch angle due to body deflection could be to build a table of accelerations and reported pitch angles with entries containing correction terms to the pitch. Such a table can be constructed by first surveying a relatively flat surface area (i.e. a parking lot) while moving the vehicle at a constant speed. This constant velocity minimizes body deflection and therefore the roll and pitch data will be more accurate. Other sensors such as a laser rangefinder can be used to create such surface maps. Using this map, and the accurate two dimensional (x, y, ϕ) data, it is possible to guess the correct roll and pitch angle. By comparing this estimate with the reported roll and pitch while accelerating forward and laterally, a correction term for each angle can be determined and stored in a table indexed by the corresponding acceleration. This means that during normal vehicle operation, this table provides a correction factor to the roll and pitch angles given the lateral and forward acceleration data. More investigation is necessary to study the nature of these correction factors and their variance with steepness of the road.

One major problem is that the estimate of positioning depends on pitch and roll which could be erroneous. This brings about some uncertainty in the coordinates of the vehicle as it moves. As the vehicle returns to its original position, the sensors erroneously report a different position because of accumulated errors partially cause by inaccurate roll and pitch angles.

If the goal is the recreation of a motion profile without stressing the accuracy of the positioning data with respect to the world, effects of roll and pitch angles could be ignored in two dimensional position estimation. This means that, while the data is incorrect with respect to the world, it is consistent with respect to the vehicle. In other words, although the data is incorrect, the vehicle will report the same data when it reaches the same position in the world assuming that the errors were primarily due to roll and pitch angles.

Therefore, two dimensional positioning is used for map following and tracking while the three-dimensional data is maintained separately for sensor positioning and calibration.

Section 3

The Clients

Assuming that the robot is aware of its location, its next challenge is deciding how to maneuver itself to achieve a goal position or motion profile. As previously mentioned, questions such as "Where do I want to go?" or "How do I get there?" are resolved by the perception or mapping clients. The two classes of clients are the navigators and the observers. The Navlab controller has three internal clients: the mapper, the tracker, and the velocity regulator. The rest of the clients are servers for other hardware or software clients external to the controller. These servers mimic the presence of the external clients inside the controller by using a high-speed communication network. Figure 3.1 shows the structure of the clients for the Navlab controller.

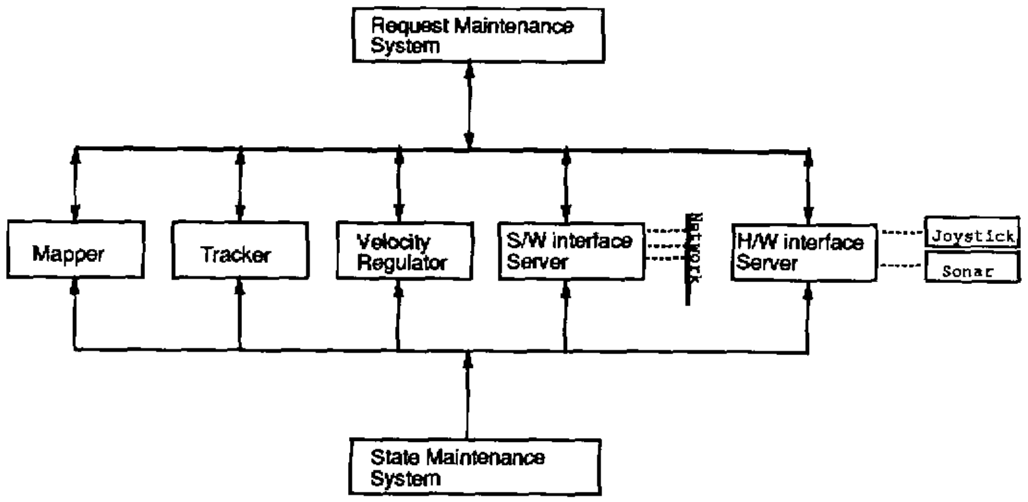

Figure 3.1: Clients of the Navlab Controller

3.1 Path Specification

In previous systems on the Navlab, the steering loop was closed through external sensor modules. The controller was given a series of circular arcs to follow. The actual path tracked was inaccurate because transitions from one arc to the next could not be made instantaneously. Since there was little feedback to higher lever modules, they assumed perfect path tracking, which led to errors in their world models, which in turn increased their computational load.

The perception and mapper modules are responsible for determining a suitable path to follow based on mapped or sensed data and informing the low-level controller of their decisions. The issues here are how does this flow of information take place and who does what. Past control used a "Virtual Vehicle Interface" [6] to specify motion as

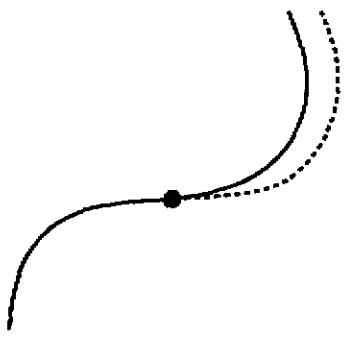

Figure 3.2: Effect of instantaneous change in curvature

a series of arcs which the controller simply translated to a steering wheel position. This was done at the expense of more complex processing by every high-level module in the system.

Another factor to consider is the effect of roll and pitch angle to steering the vehicle. For example, in a banked corner, the steering angle is smaller for the same radius of curvature from a flat surface.

The experience with these systems led to the decision of moving all the path servoing to the low-level controller, freeing the navigator clients from this task. This approach allows experimenting with different tracking strategies to yield desirable features such as continuous curvature and smooth motion without major changes in the rest of the system. Taking this approach, a desired path planned by the perception modules is passed to the controller as a set of two-dimensional world coordinate points along with the desired velocity to move the vehicle. The controller interpolates between the supplied points to estimate a heading and curvature at the given points for use by the tracking algorithms. This means that the points should be in fairly small intervals to estimate the path as closely as possible. This approach is compatible with the old system since arcs can be sampled to give the same information.

3.2 The Mapper

The mapper's main responsibility is keeping the list of points specifying the requested path as consistent as possible. The requested path, represented by a list of (x, y) coordinates is available to the mapper through the request maintenance system. The mapper connects the path points with straight lines and estimates the heading of the mid-point of these line segments based on the slope of the line and averages these values to estimate a heading at each path point. The difference in the heading from one mid-point to the next is used to compute the curvature at each path point. The heading and curvature data is necessary for certain tracking algorithms. Figure 3.3 is an exaggerated view of a list of three path points. To ensure the accuracy of the calculated heading and curvatures, the list of points supplied to the mapper must be in small distance intervals.

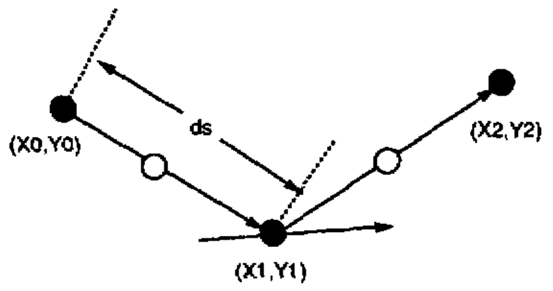

Figure 3.3: A list of path points supplied to the mapper

$$\phi_1 = \frac{1}{2}(atan2[(X_1 - X_2),(Y_2 - Y_1)] + atan2[(X_0 - X_1),(Y_1 - Y_0)])$$
$$\gamma_1 = \frac{1}{ds}(atan2[(X_1 - X_2),(Y_2 - Y_1)] - atan2[(X_0 - X_1),(Y_1 - Y_0)])$$

The mapper also keeps the points consistent if the navigators change the desired path. All points in the front of the vehicle are replaced by the most recent set of path points and all the necessary data recomputed.

The mapper can be configured to record a motion profile as the vehicle moves to produce a map of the traveled path. In this instance the actual heading and curvature from the state maintenance system are recorded. The information recorded at each point is $(x, y, z, \psi, \theta, \phi, \gamma)$. These points are recorded at regular distance intervals small enough to represent a map. The length of this interval depends on the minimum radius of curvature of the vehicle and is inversely proportional to the sharpness of the path. For example, the Navlab's minimum turning radius is 7 meters and paths can be accurately represented by recording points every half meter.

The mapper stores this map as a separate entity and can supply the tracker with points from this prestored map for blind path tracking. The mapper can be configured to refer to its prestored map by other clients at any time. This means that a new system undergoing tests can tell to the mapper that it needs assistance and the mapper uses the prestored map instead of the client's specified path.

3.3 The Tracker

The mapper's maintained set of path points (x, y, ϕ, γ) are used by the tracker in conjunction with the current vehicle status to continuously command a new steering angle which would keep the vehicle on the desired path. The tracker reexamines the path every time it cycles enabling the mapper to vary the path as the navigators make adjustments to their previously requested path points. The basic tracking strategy continuously executes the following steps:

1. Choose a point somewhere ahead of the vehicle on the desired path as the goal point.
2. Use the information at the goal point to define a path to follow.
3. Using the defined path, calculate a new steering angle until it is time for a new goal point

The distance between the vehicle and the goal point is called the "lookahead" distance. Suitable lookahead distances should be chosen for different speeds. If the lookahead is too long, the vehicle may cut corners and if too short, oscillations may result. The exact relationship between the lookahead and speed requires more investigation. Depending of the tracking approach taken, suitable lookahead distances can be determined experimentally for different speeds to discover the relationship of lookahead and speed. Generally, a longer lookahead is desired for higher speeds. For the Navlab, the lookahead distance is typically between 5 to 15 meters.

Details of choosing a lookahead distance, defining a path, and deciding when to reselect a goal point are specific to the tracking strategy. Three alternate strategies are applied in the Navlab tracker. The first is a control theory approach using errors in (x, ϕ), the second attempts to fit a suitable polynomial to the desired point, and the third uses successive arcs to reach the goal point.

3.3.1 Control Theory Approach

Consider a goal point a lookahead distance away as shown by Figure 3.4. The goal point has a negative x coordinate and and zero heading in the vehicle's coordinate frame.

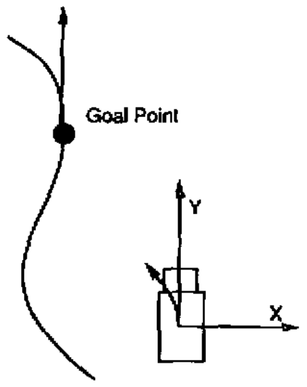

Figure 3.4: Compensation for error in X

Intuitively, to reach this point, the steering wheel must be turned counter clockwise by a certain amount which represents a positive change in steering angle.

Now, consider a positive heading but a zero x coordinate for the goal point as depicted by Figure 3.5. Again, to correct for the heading error, the steering wheel must be turned counter clockwise by a certain amount causing a positive change in the steering angle.

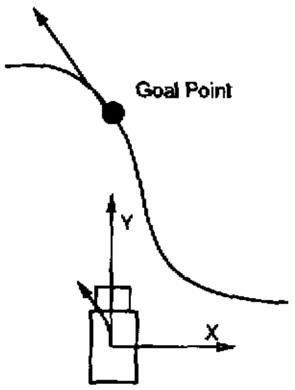

Figure 3.5: Compensation for error in Heading

Therefore, a positive error in x must contribute a negative component to the steering angle and a positive error in heading must increase the steering angle. To better understand this tracking strategy, examine Figure 3.6 that shows the situation with a straight path.

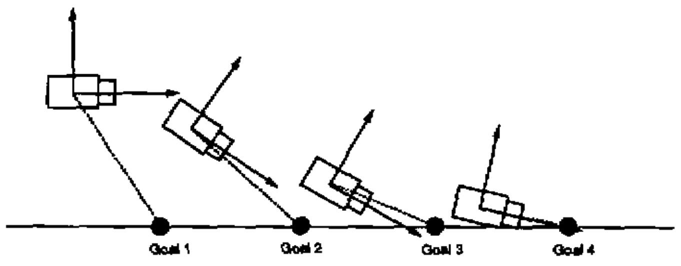

Figure 3.6: Incremental goal points of the Control Theory tracker

The first goal point has a large positive x error but no heading error. This positive x error moves the steering wheel sharply clockwise. As the vehicle moves, the heading error increases, the x error decreases causing the steering wheel to start moving counter clockwise. As the vehicle approaches the path, x and heading error become small and the steering angle becomes smaller and smaller in magnitude.

Using this intuition a new curvature (γ') may be computed by:

$$\gamma' = g_1 \Delta\phi - g_2 \Delta x \tag{3.1}$$

where

$$\begin{aligned} \Delta\phi &= \phi_{goal} - \phi_{now} \\ \Delta x &= x_{goal} - x_{now} \end{aligned}$$

The gains g_1 and g_2 are parameters related to system's performance. It would be beneficial to find the relationship between these two gains in order to tune them more efficiently. Recall:

$$\sin(\phi) = -\frac{dx}{ds} \tag{3.2}$$

Assuming the change in heading to the goal point is small, it is possible to use the small angle approximation to approximate ϕ.

$$\phi = -\frac{dx}{ds} \tag{3.3}$$

therefore defining γ' as a function of distance (s):

$$\gamma'(s) = \frac{d\phi}{ds}(s) = -\frac{d^2x}{ds^2}(s) \tag{3.4}$$

from equation 3.1:

$$\gamma'(s) = g_1(\phi_{goal} - \phi(s)) - g_2(x_{goal} - x(s)) \tag{3.5}$$

Rewriting this equation gives:

$$\begin{aligned} \frac{d\phi}{ds} &= (g_1\phi_{goal} - g_2 x_{goal}) - g_1\phi(s) + g_2 x(s) & (3.6) \\ -\frac{d^2x}{ds^2}(s) &= (g_1\phi_{goal} - g_2 x_{goal}) + g_1\frac{dx}{ds}(s) + g_2 x(s) & (3.7) \\ & & (3.8) \end{aligned}$$

Rearranging leads to the following differential equation:

$$\frac{d^2x}{ds^2}(s) + g_1\frac{dx}{ds}(s) + g_2 x(s) + (g_1\phi_{goal} + g_2 x_{goal}) = 0 \tag{3.9}$$

Therefore, by deciding whether the system should be over-damped or slightly under-damped it is possible to find g_2 once g_1 is tuned experimentally. if g_2 is chosen such that $g_1 > 2\sqrt{g_2}$ the system will be overdamped.

Figure 3.7 displays the performance of this tracking strategy. This map was chosen because of its particularly sharp curves, especially the one on the bottom right of the figure, which is almost at the minimum turning radius for the Navlab.

Considering the simplicity of this tracking method, the performance is surprisingly good. The above results were obtained by this tracker cycling 10 times per second. The gain g_1 was chosen by observing the oscillations of the steering wheel and tracking performance. The tracking performance was reasonable at speeds less than 3 meters per second. Due to slop in the steering mechanism, attempts for overcoming small heading errors led to small oscillations of the steering wheel which are not desirable in high speeds. Better tracking strategies are necessary to overcome this problem.

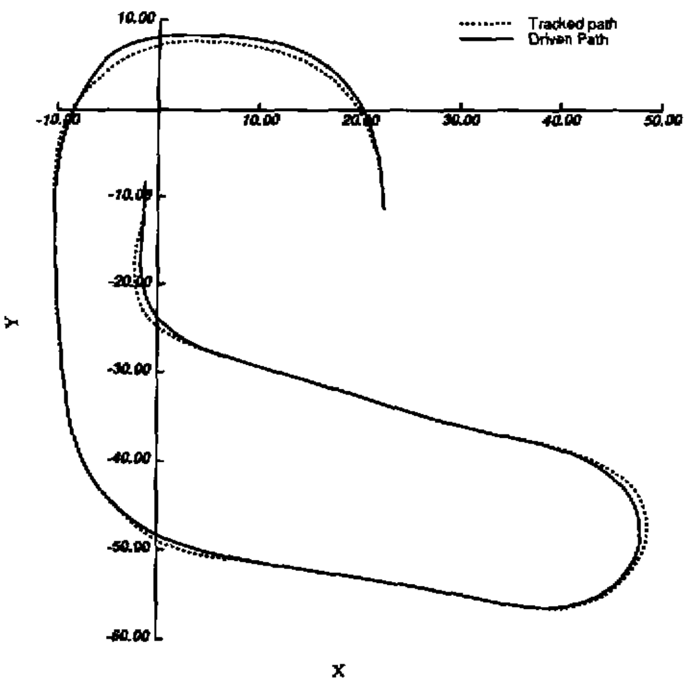

Figure 3.7: Control Theory tracker performance at 2.5 m/s

3.3.2 Quintic Polynomial Fit

One way to avoid the discontinuous motion of the steering wheel exhibited by the control theory approach is to fit a polynomial that originates at the vehicle and reaches the goal point meeting the position, heading, and curvature constraints at both points.

Transforming the goal point to the vehicle coordinate frame, we have the following constraints at the end points:

$$\begin{aligned} X_{origin} &= 0 \iff X_{goal} = X_L \\ Y_{origin} &= 0 \iff Y_{goal} = Y_L \\ \phi_{origin} &= 0 \iff \phi_{goal} = \phi_L \\ \gamma_{origin} &= \gamma_0 \iff \gamma_{goal} = \gamma_L \end{aligned}$$

Consider a parameter (t) varying from 0 at the origin to 1 at the goal point. Given that x and y and their first and second derivatives are available, connecting these two points and meeting the six constraints requires that x and y be fifth order polynomials in t.

$$\begin{aligned} x\ (t) &= a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \\ \dot{x}\ (t) &= a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \\ \ddot{x}\ (t) &= 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \end{aligned}$$

$$\begin{aligned} y\ (t) &= b_0 + b_1 t + b_2 t^2 + b_3 t^3 + b_4 t^4 + b_5 t^5 \\ \dot{y}\ (t) &= b_1 + 2b_2 t + 3b_3 t^2 + 4b_4 t^3 + 5b_5 t^4 \end{aligned}$$

$$\ddot{y}\ \ (t) = \ 2b_2 + 6b_3t + 12b_4t^2 + 20b_5t^3$$

Therefore, to fit these quintic polynomials, the available data at the end points must be used to compute the first and second derivatives of x and y. Recall:

$$\begin{aligned} \frac{dx}{ds} &= -\sin(\phi) \\ \frac{dy}{ds} &= \cos(\phi) \end{aligned}$$

which means:

$$\begin{aligned} \frac{d^2x}{ds^2} &= -\cos(\phi)\frac{d\phi}{ds} \\ \frac{d^2y}{ds^2} &= -\sin(\phi)\frac{d\phi}{ds} \end{aligned}$$

Using the chain rule:

$$\begin{aligned} \dot{x} &= \left(\frac{dx}{ds}\right)\left(\frac{ds}{dt}\right) = -\sin(\phi)\left(\frac{ds}{dt}\right) \\ \dot{y} &= \left(\frac{dy}{ds}\right)\left(\frac{ds}{dt}\right) = \cos(\phi)\left(\frac{ds}{dt}\right) \\ \ddot{x} &= \frac{d^2x}{ds^2}\left(\frac{ds}{dt}\right)^2 = -\cos(\phi)\left(\frac{d\phi}{ds}\right)\left(\frac{ds}{dt}\right)^2 \\ \ddot{y} &= \frac{d^2y}{ds^2}\left(\frac{ds}{dt}\right)^2 = -\sin(\phi)\left(\frac{d\phi}{ds}\right)\left(\frac{ds}{dt}\right)^2 \end{aligned}$$

Assuming that the vehicle travels at a constant speed, $\frac{ds}{dt} = L$ where L is the length of the quintic curve, the boundary conditions are related to the available data by:

$$\begin{aligned} \dot{x} &= -L\ \sin(\phi) \\ \dot{y} &= L\cos(\phi) \end{aligned}$$

$$\begin{aligned} \ddot{x} &= -L^2\cos(\phi)\left(\frac{d\phi}{ds}\right) \\ \ddot{y} &= -L^2\sin(\phi)\left(\frac{d\phi}{ds}\right) \end{aligned}$$

It seems that the three boundary conditions $(x, \dot{x}, \ddot{x})$ and $(y, \dot{y}, \ddot{y})$ at both points are available. The only problem is determination of the constant L which is $(\frac{ds}{dt})$. This means that the length of the curve is necessary in computing the polynomial representing it. Evaluating the equations at the end points gives the following results:

At the origin:

$$\begin{aligned} x(0) = 0 &\rightarrow a_0 = 0 \\ y(0) = 0 &\rightarrow b_0 = 0 \end{aligned}$$

$$\begin{aligned} \dot{x}(0) = -L\ \sin(0) &\rightarrow a_1 = 0 \\ \dot{y}(0) = \ L\cos(0) &\rightarrow b_1 = L \end{aligned}$$

$$\begin{aligned} \ddot{x}(0) = -L^2\cos(0)\left(\frac{d\phi}{ds}(0)\right) = 2a_2 &\rightarrow a_2 = -\frac{L^2\,\gamma_0}{2} \\ \ddot{y}(0) = L^2\sin(0)\ \gamma_0 &\rightarrow b2 = 0 \end{aligned}$$

At the goal point:

$$\begin{aligned} x_{goal} &= -\frac{L^2\,\gamma_0}{2} + a_3 + a_4 + a_5 \\ y_{goal} &= L + b_3 + b_4 + a_5 \end{aligned}$$

$$\begin{aligned} -L\ \sin(\phi_{goal}) &= -L^2\gamma_0 + 3a_3 + 4a_4 + 5a_5 \\ L\ \cos(\phi_{goal}) &= 2L + 3b_3 + 4b_4 + 5b_5 \end{aligned}$$

$$\begin{aligned} -L^2\gamma_{goal}\ \cos(\phi_{goal}) &= -L^2\gamma_0 + 6a_3 + 12a_4 + 20a_5 \\ -L^2\gamma_{goal}\ \sin(\phi_{goal}) &= 2L + 6b_3 + 12b_4 + 20b_5 \end{aligned}$$

Iterative methods can be used to compute the constants and L simultaneously. In one attempt, L was initially estimated as its minimum length which is the distance between the vehicle and the goal point. At each iteration, the constants were computed using the guessed L. Using the computed constants, sines and cosines of the heading were computed at discrete points on the quintics. If all of the sine and cosine values were in the correct range (i.e. -1 to 1), the quintics were accepted. Otherwise, L was increased by a fraction of its previous value and the cycle was repeated. This approach had limited success and a high computational cost.

A simpler approach to this problem is to assume that the heading at the goal point in vehicle coordinates is less than $\frac{\pi}{2}$. This enables us to describe x as a quintic polynomial in y and available data can be used without the knowledge of the length L.

$$\begin{aligned} x\ \ (y) &= a_0 + a_1y + a_2y^2 + a_3y^3 + a_4y^4 + a_5y^5 \\ \dot{x}\ \ (y) &= a_1 + 2a_2y + 3a_3y^2 + 4a_4y^3 + 5a_5y^4 \\ \ddot{x}\ \ (y) &= 2a_2 + 6a_3y + 12a_4y^2 + 20a_5y^3 \end{aligned}$$

The derivatives of x with respect to y are:

$$\frac{dx}{dy} = -\tan(\phi)$$

$$\frac{d^2x}{dy^2} = -\sec^2(\phi)\left(\frac{d\phi}{dy}\right)$$

Using the chain rule:

$$\begin{aligned}\frac{d\phi}{dy} &= \left(\frac{d\phi}{ds}\right)\left(\frac{ds}{dy}\right)\\ \text{or } \frac{d\phi}{dy} &= \left(\frac{d\phi}{ds}\right)\sec(\phi)\end{aligned}$$

Finally, this leads to:

$$\begin{aligned}\frac{dx}{dy} &= -\tan(\phi)\\ \frac{d^2x}{dy^2} &= -\gamma\ \sec^3(\phi)\end{aligned}$$

To simplify computations scale x and y so that y varies from 0 to 1. Examining the boundary conditions, at the origin:

$$\begin{aligned}x(0) = 0 &\rightarrow a_0 = 0\\ \dot{x}(0) = 0 &\rightarrow a_1 = 0\\ \ddot{x}(0) = \gamma_0 &\rightarrow a_2 = -\gamma_0/2\end{aligned}$$

at the goal point:

$$\begin{aligned}a_3 + a_4 + a_5 &= x - a_2 \equiv k_1\\ 3a_3 + 4a_4 + 5a_5 &= -\tan(\phi_{goal}) - 2a_2 \equiv k_2\\ 5a_3 + 12a_4 + 20a_5 &= -\gamma_{goal}\sec^3(\phi_{goal}) - 2a_2 \equiv k_3\end{aligned}$$

solving for a_3, a_4, and a_5 :

$$\begin{aligned}a_3 &= 10k_1 - 4k_2 + \frac{1}{2}k_3\\ a_4 &= -15k_1 + 7k_2 - k_3\\ a_5 &= 6k_1 - 3k_2 + \frac{1}{2}k_3\end{aligned}$$

To track paths, using the above results, a quintic is fitted to the goal point and a new curvature is computed based on the y coordinate of the vehicle. Due to the vehicle's imperfect execution of the commanded curvature a quintic is only followed for a short distance before another one is fitted to a new goal point a distance ahead of the previous goal point and the process is repeated.

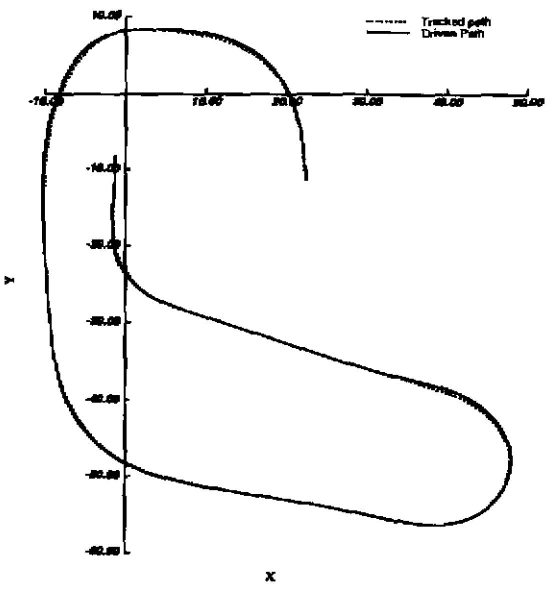

Figure 3.8: Tracking with quintics at 3.0 m/s (simulation)

Figure 3.8 shows the performance of this tracker for the previously used path. Although good results were obtained both in simulation and the real vehicle, the issues causing some complications were:

- It was unclear how far to travel on a quintic before computing a new one. Since the curvature is continuous at the origin of the quintic, cycling too fast will cause little or no changes in the curvature. On the other hand, cycling slowly and traveling on one quintic for a longer distance may cause drifts off the path due to imperfect actuation.
- The tracker's performance was very dependent on the lookahead distance. One reason for this is that the vehicle's kinematic constraints (i.e. minimum turning radius) must be met throughout the quintic and depending on the type of path followed, longer or shorter lookahead distances were required to have the optimum quintic curve.

The complexity and the ill-understood issues of this tracking approach limited its practicality and tracking results were not consistently good.

3.3.3 Pure Pursuit

The inconsistent performance of the quintic polynomial tracker motivated yet another approach to the tracking problem which resulted in the most stable and accurate tracker for the Navlab. In the quintic approach, in order to account for the changes in the path ahead, different polynomials were fit to different goal points as the vehicle traveled forward. This meant that the entire length of any one polynomial was never completed, which implies that meeting curvature and heading constraints at the goal points is not very critical. Wallace et al. [14] successfully followed roads by keeping the road image seen by the robot's cameras centered in the image. They used the displacement of the road centerline from the center of forward field of view of the cameras to compute a new steering angle. Analysis of this method shows that is it a "Pure Pursuit" strategy.

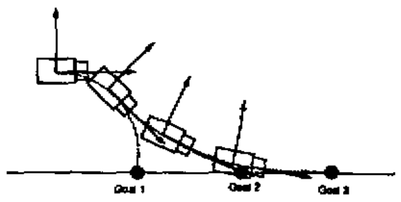

Figure 3.9: A sequence of Pure Pursuit cycles

Instead of a complex polynomial, the x and y displacements are used to fit a circular arc to the goal point. Therefore the path is tracked by repeatedly fitting different arcs to different goal points as the vehicle moves forward. Figure 3.9 illustrates a sequence of tracking cycles. Figure 3.10 shows how the arc fitting is performed.

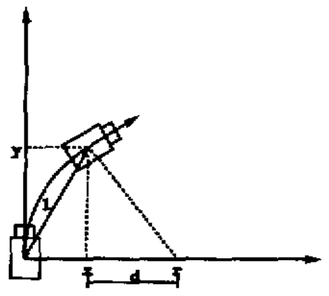

Figure 3.10: Fitting and arc for Pure Pursuit

Using the following equations, r can be determined in terms of x and y.

$$x + d = r \quad \rightarrow \quad d = r - x$$
$$d^2 + y^2 = r^2 \quad \rightarrow \quad (r - x)^2 + y^2 = r^2$$

solving for r,

$$r = \frac{x^2 + y^2}{2x} \tag{3.10}$$

Therefore, for each goal point a constant lookahead (l) distance away, the calculated curvature $\gamma' = \frac{2x}{x^2+y^2}$. Cycling this tracker at 10-15 times a seconds produced smooth steering wheel movements and gave extremely good results on the Navlab with speeds up to 8 m/s (18 MPH) which is its top speed. Figure 3.11 shows the performance of this tracker following the same challenging map the quintic tracker followed.

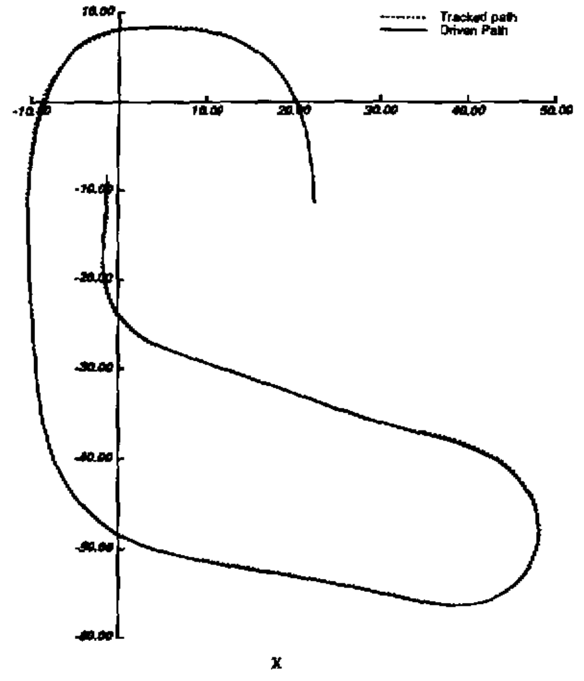

Figure 3.11: Pure Pursuit tracker performance at 3.5 m/s

Noting that $l^2 = x^2 + y^2$, equation 3.10 can be rewritten as,

$$r = (\frac{l^2}{2x}) \tag{3.11}$$

which means that:

$$\gamma = (\frac{2}{l^2})x \tag{3.12}$$

Thus Pure Pursuit is simply a proportional controller of the steering angle using the x displacement as the error and $(\frac{2}{l^2})$ as the gain. Reasons for its high performance could be attributed to the ease of experimental tuning of the gain by simply changing the lookahead distance and to the absence of noisy derivative terms (heading, curvature) in its servo loop. Overall, this tracking strategy led to the most stable and accurate path tracking for the Navlab.

3.4 Velocity Regulator

Experience has shown that extremely accurate velocity control is not very important as long as positioning data is maintained very accurately to report the actual behavior of the vehicle. Instead of accuracy, a smooth velocity profile is very desirable for the Navlab considering its six ton weight and the equipment on board. The main issue is to achieve the desired velocity without stalling the engine, spinning the wheels, or tossing the occupants around.

This type of decision making is a level higher than the low-level velocity loop that moves mechanical components to increase or decrease the velocity. For example, if the vehicle is traveling uphill, a request for increase in velocity will cause the low-level velocity loop to further increase the load on the engine. But if the engine is unable to deliver the requested torque, the velocity loop will keep pushing for the requested velocity and the engine will start stalling, further decreasing its torque output. This in turn will feed back and the velocity loop will push even harder to achieve the requested velocity, and this quick sequence of events will stall the engine. Therefore, a mechanism is necessary to observe vehicle behavior to limit this request for torque or change the load gradually if the engine is unable to deliver or it is undesirable.

In situations where the wheels are slipping, the torque output must be decreased to the point where there is enough traction to enable the vehicle to move. One approach is to record the performance of the vehicle in different situations and determine limits for acceleration and rate of change of acceleration to ensure desirable behavior. Factors such as vehicle pitch and roll must be taken into account. For example, these limits may vary while going up or down hill.

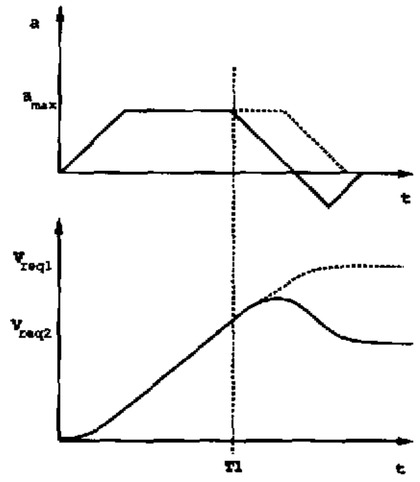

Figure 3.12: Effect of reducing the requested Velocity

A typical PID closed loop velocity controller requires a requested velocity and direction of motion to produce a torque output. More sophisticated ones reach the velocity linearly if an acceleration is supplied to them. The following is pseudo-code of the Navlab's velocity regulator client:

```
If ($V_{req}$ is in different direction from $V_{cur}$) then $V_{req} = 0$
$\tau = | \frac{a}{\alpha} |$
$V_\beta = V_{curr} + a(\frac{\tau}{2})$
if ($V_{req} == V_\beta$) then $a_{dir} = -1$
if ($V_{req} < V_\beta$) then $a_{dir} = -1$
if ($V_req > V_\beta$) then $a_{dir} = +1$
```

Where:

a	$\equiv$	acceleration
a_{dir}	$\equiv$	direction of change acceleration
α	$\equiv$	rate of change of acceleration
V_{cur}	$\equiv$	current velocity
V_{req}	$\equiv$	requested velocity
V_β	$\equiv$	velocity reached if a is reduced to zero
τ	$\equiv$	time it takes for a to reach zero

If V_{req} is in a different direction from V_{cur}, the goal should be to stop the vehicle before movement in the opposite direction. At any time, letting the acceleration a increase or decrease to zero will take τ seconds during which the velocity will change to V_β. The variable, a_{dir}, is used to specify an increase or decrease of a in time with $| a |$ limited to a maximum safe value a_{max}.

For example, suppose a positive velocity is requested which is higher than V_{cur} and ($a > 0$). If V_{req} is equal to V_β, it will be achieved if a is ramped down immediately ($a_{dir} = -1$). But if V_{req} is less than V_β, it is simply not possible to achieve it before reaching V_β and the best that can be done is to overshoot the requested velocity up to V_β and decelerate at that point. Figure 3.12 illustrates the velocity profile when the requested velocity was reduced from V_{req1} to V_{req2} at time T_1.

On the other hand, if V_{req} is greater than V_β, then a must be ramped up (i.e. $a_{dir} = +1$) which means an increase in a or no change if a is at its limit already.

The above scheme works for any value of a, that is regardless of accelerating or decelerating, provided that V_{req} and V_{cur} are in the same direction which is guaranteed by the first conditional statement.

3.5 Software Interface Server

The most difficult problems in mobile robot control arise in interpreting image and range data in order to plan paths and avoid obstacles. Because of their complexity, it is desirable to run the clients in charge of these tasks on powerful computers external to the controller while keeping them as one of the clients. The Software Interface Server is the communication link that provides this capability.

The server allows multiple clients to connect and disconnect dynamically by reserving one connection to listen for new clients in need of service. Each connection to the network has a name and a priority which are determined at connecting time. For example, an obstacle avoidance system has priority over a road follower. The server wakes up when there are pending requests and it takes a snapshot of the clients in need, and services all of them based on their priority before reexamining the connections.

Although, it has never been necessary for the systems on the Navlab, the system has provisions for multiple software interface servers. One reason for this is to have more connections than the limit per server. But more importantly, this is a good way of separating essential and non-essential requests. For example, with one server an unimportant status request from an obstacle detector is serviced before a potentially critical motion request from a road follower.

The server is invisible to the external modules. All of their requests are relayed through the network by invocation of procedure calls. These procedures build data packets and transfer them to the controller and if necessary extract requested data from received packets from the interface server. All the packets contain text since the packet sizes are typically very small and the penalty of extracting the data from text was insignificant compared to the flexibility of communication to any other computer regardless of data representation details such as byte ordering. For fast communication the following steps were taken:

- There is no attention given to data integrity since the protocols typically used for data transfer across the Ethernet (i.e. TCP/IP) guarantee data integrity.
- All data packets have the same size and start with an opcode integer which is an index in a command table. This eliminates speed penalties as the number of opcodes increases.
- Every command table entry contains the number of parameters associated with the opcode and the address of the service routine to be invoked to process the command.

Unlike general interfaces such as the Virtual Vehicle Interface, the commands are tailor made to the requirements of the vehicle. The process of creating new commands was facilitated by a software library that inserts new opcode entries in the command table provided the service routine is supplied by the user. For example, an external mapping and tracking module may require status information and send path points in one command, enabling it to pipeline its operation. This specific message configuration can be handled by writing a service routine which accesses and updates data in the controller. The addition of this new command is completed by dynamically linking it with the rest of the modules in the controller.

All of these measures improved the speed and therefore the accuracy of the interface. The average communication delay for the implemented system on board the Navlab is 3 milliseconds for one-way commands (i.e. no reply packet from the controller) and about 14 milliseconds for two-way commands (i.e. status queries in addition to a one-way command). These delay times were measured with three external modules requesting data from the controller simultaneously.

3.6 Hardware Interface Server

The dual of the software interface server is the hardware interface server. The clients of this server are certain pieces of hardware directly involved in vehicle control. An example of such hardware is a sonar bumper reporting signals as obstacles enter a monitored zone around the vehicle.

Human interaction tools such as a joystick are other examples of clients of this server. The Navlab's joystick can be used to precisely maneuver the vehicle. The joystick becomes enabled by squeezing the trigger or dead-man switch and moves the steering wheel as it is moved left and right. The speed of the steering wheel actuator is regulated by the difference in the current steering wheel position and the requested position by the joystick. This allows the joystick to have a smooth response with small hand movements with out penalizing performance for sudden large changes that are critical if the vehicle is out of control. The back and forth movement of the joystick simply translates to a requested velocity which is correctly handled by the velocity regulator. The main contribution of the joystick has been the ability to merge computer and joystick control to correct minor deviations off the desired path in cases of miscalibrated sensors or system parameters. If these deviations become very large, it is possible to override the computer signals by double clicking the dead-man switch. This human interface has been proven valuable in configuring new systems. The joystick also enables the user to speed up or slow down the vehicle during a test run to quickly determine the system's performance with different vehicle speeds.

Section 4

Actuation System, Watch Dog, and Request Maintenance System

4.1 Actuation System

The Navlab's Actuation System encapsulates the details of the hardware controllers and actuators.[1] Figure 4.1 shows the structure of this system.

The low-level servo loops that control the position of the steering wheel and velocity of the drive shaft are both closed in the actuation system. This allows the higher level system clients to concentrate on deciding upon a speed and a steering angle rather than these hardware control issues. A "Hardware Configurator" process is constantly cycling and delivering consistent requests to all hardware controllers which in turn move the actuators.

4.1.1 Steering Actuation

The steering wheel is rotated by a DC servo motor. A high precision (2400-line) optical encoder mounted on the steering column supplies data to an off-the-shelf Creonics [1] motion controller. The signal produced by this hardware controller is amplified and sent to the DC motor to produce motion. Therefore, provided that the PID gains of the Creonics card are correctly tuned, a requested steering wheel position is a single number which is calculated by the requested steering angle and the previously determined ratio of steering angle and encoder position of the steering column.

The PID controller's gains were tuned experimentally. Since the vehicle trajectory is controlled by the steering wheel, the steering wheel's accurate positioning is critical. To provide damping, the differential gain was increased to the point that motor movement was smooth as the shaft was manually turned. Maintaining this gain, the proportional gain was tuned by generating a step input to the motor and observing its behavior. This gain was increased to the highest value that did not exhibit "ringing" or oscillations. A small integral gain was experimentally determined to improve the static positioning accuracy. The smallest gain that achieved the desired result was chosen since a high integral gain causes instability.

4.1.2 Drive Actuation

Navlab is propelled by a hydraulic transmission of power from the engine to the wheels. The engine, running at a constant speed, drives a hydraulic pump supplying fluid to the two speed transmission through an electrically controlled valve. The opening of the valve depends on the current supplied to the valve actuator. Similar to the steering motor's input, this current is regulated by the control signal generated by a Creonics PID controller. A high voltage from the

[1]see [4] for more details

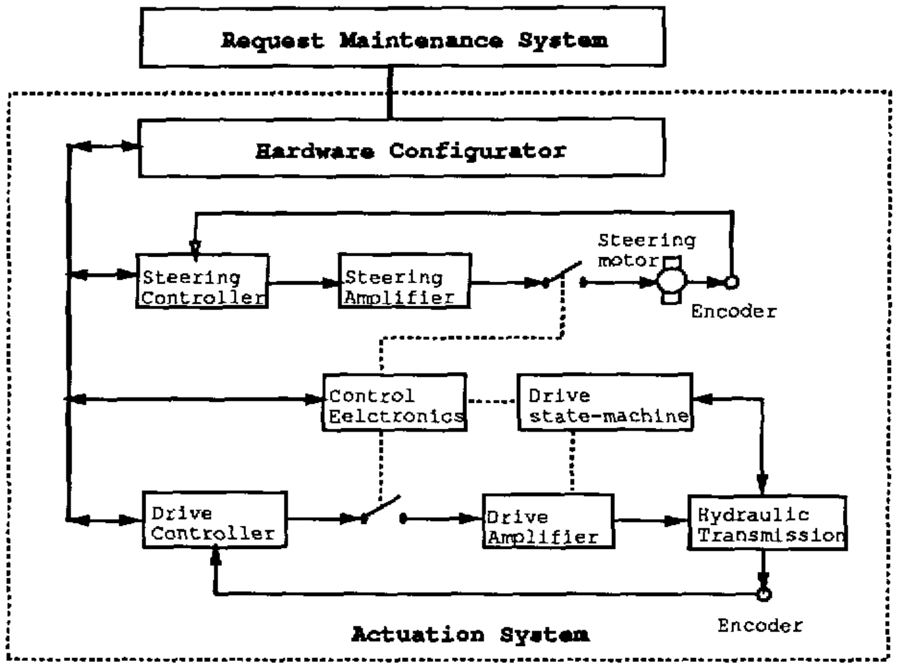

Figure 4.1: Block diagram of the Actuation System

motion controller is translated to high current and opens the valve in a direction that causes forward motion. Similarly, to stop the vehicle, the valve is closed by zeroing the current and to move backwards, current is supplied in the negative direction.

The gains of this controller were tuned experimentally as well. Considering the weight of the vehicle with the equipment and researchers on board, more attention was given to smoothness of operation rather than high accuracy of motion.

The drive state of the transmission is maintained by a hardware state machine that is closely coupled to the state configurator. The information from this state machine is stored in the state maintenance system as well. The hardware configurator uses this information to change the PID gains or encoder ratios if necessary. For example, the distance traveled per shaft rotation and system response is different when the transmission is engaged in different gears.

4.2 The Watch Dog

The watch dog is responsible for the health of the vehicle. It is in parallel to the decision makers and it can choose to override their decisions if they are out of the range of vehicle performance or simply considered undesirable. For example, the duties of the watch dog of the Navlab controller are:

- not allowing a lateral acceleration of more than 0.1 g. This is done by interfering to slow down the vehicle in sharp curves.
- examining the drive state of the transmission and updating the state maintenance system to reflect the correct information as the vehicle is engaged in computer controlled or manual mode.
- stopping the vehicle if the vehicle pitch is more than 30 degrees. This is because the hydraulics can not stop or move the vehicle on roads this steep.

- blowing the horn if any of the control systems fail. For example, if there is an amplifier failure that would disable the steering motor, recognizing the seriousness of this situation, the horn is continuously blown until the vehicle is reengaged in manual mode. Also, by different horn patterns, the watch dog signals the failure of different on board sensors.
- examining memory locations that are regularly updated by all of the processes to determine whether a process has failed to cycle. Although not yet available, to detect the failure of the watch dog itself, a piece of hardware is necessary that will sound a warning if a hardware line is not reset at regular intervals by the watch dog.
- printing diagnostic messages to a console and storing them in a file for future examination. The watch dog can also be configured to display different system parameters continuously for monitoring purposes.

4.3 Request Maintenance System

The Request Maintenance System is the communication link between the clients, the watch dog, and the actuation system. It handles the bookkeeping of incoming and outgoing requests. All incoming and outgoing data is stored in one central buffer. Mutual exclusion issues are handled by reducing the size of data reads and writes so that they are atomic operations. This keeps the number of semaphores to a minimum to improve performance. The "Request Processor" continuously monitors the incoming requests and updates the request buffer with outgoing requests. Its main functions are:

- Receiving the requested path points from all clients and sending the requests of the highest priority client to the mapper.
- Sending the turning radius and velocity generated by the tracker to the actuation system.
- Merging the requested steering angles and velocities from the hardware clients such as the joystick, with the ones from the tracker. It also overrides the tracker requests if the joystick is in override mode.
- Directing the watch dog to make various changes to the state buffer inside the state maintenance system. A few of these requests are: changing the positioning data upon recognition of a landmark by a perception client; changing the mode of the reported position data to dead reckoned, 3-D, or 2-D positioning; changing the lookahead distance or the tracking algorithm used by the tracker.

Section 5

Conclusions

The conclusions of this report are:

- The vehicle state and positioning information should be maintained by the low-level real-time controller.
- Data from multiple sources (INS, GPS, dead reckoning) must be merged to produce one accurate list of status information sharable by all systems on board the vehicle.
- Positioning using an INS or dead reckoning is not free of drift and a higher level landmark recognition system is necessary to periodically compensate for this drift.
- The low-level controller must support multiple clients both internally and externally. Available high speed networks and server configurations are able to meet the timing requirements of most external clients.
- Path tracking must be an internal feature of the low-level controller in order to free all other clients from this task.
- Excellent tracking results were obtained using a "Pure Pursuit" tracker cycling 10 to 15 times per second.
- Separating the servo loops that control the actuators from the rest of the system greatly facilitated the implementation of complex control strategies specific to the vehicle. (i.e. smooth velocity, merging of steering control)
- Details of actuation and system bookkeeping must be isolated from the rest of the system in order to keep the control architecture reusable.

These conclusions were reached by implementing[1] the described integrated controller which has been successfully running the Navlab for the past six months.

5.1 Acknowledgements

I would like to thank: my advisors Takeo Kanade, Chuck Thorpe, and Pradeep Khosla for their guidance and support, Dan Christian for his endlessly valuable technical advice, Dean Pomerleau for his eagerness to use my work and his driving ability, Henning Pangels for his hints about the old controller, Dong Hun Shin and for his hints about path tracking, Hans Moravec for his mathematical insight, Gary Shaffer for listening to and developing my ideas, Jim Frazier for driving the Navlab and Gerry Roston for his Creonics device driver.

[1]see appendix A for hardware details

Appendix A

Hardware Details

A.1 Main processor

The main processor used by the controller is a 25MHz Motorola 68020 based computer (MVME133XT) [8] with an 68882 floating point coprocessor. It is a VME module and because of the wide bus and high speed of this processor, one processor module was sufficient for the demands of the current system. The processing is managed by VxWorks [13] which is a commercially available realtime kernel developed by Wind River Systems Inc. This Unix-like environment allows easy system development because of the extensive C language library and Ethernet support.

A.2 Communications

The communication link to the rest of the system is through a Motorola (MVME330) [9] Ethernet controller. Support for this board is provided by Wind River Systems Inc. including most Unix networking concepts such as sockets and pipes.

The communication to the drive system is through a Motorola 128 line parallel port (MVME340A) [10] capable of generating interrupts on the VME bus depending on certain line activity. It also consists of three on board timers for regular event generation or monitoring. For example, changing gears in the transmission is simply done by driving certain lines from the VME bus. If the actuator motion is aborted externally due to a motion fault or an emergency, the port interrupts the processor and appropriate action is taken.

A.3 Low-level PID controller

The PID controller used is a VME based Creonics [1] two-axis motion control computer. It has an encoder input and a command output for each axis. Because of its digital nature, it is very easy to program and interface to any system. All controller parameters are digital values that are configured at startup or while in operation. It also has provisions for profiling and error detection in the generated motion and it is capable of deactivating the amplifiers and the drive systems by hardware and software travel limit switches.

Bibliography

[1] *VMEbus Compatible Motion Control Card.* Etna Road, Lebanon, NewHampshire 03766, 1988.

[2] Jill D. Crisman and Charles E. Thorpe. Color vision for road following. In Charles E. Thorpe, editor, *Vision and Navigation: The Carnegie Mellon Navlab*, chapter 2. Kluwer Academic Publishers, 1990.

[3] E. Dickmanns and A. Zapp. Autonomous high-speed road vehicle guidance by computer vision. In *Proc. 10th IFAC*, Munich, 1987.

[4] K. Dowling, R. Guzikowski, J. Ladd, H. Pangels, S. Singh, and W. Wittaker. Navlab: An autonomous navigation testbed. In *Vision and Navigation: The Carnegie Mellon Navlab*, chapter 12. Kluwer Academic Publishers, 1990.

[5] D. Feng. *Satisficing Feedback Strategies for the Local Navigation of Autonomous Mobile Robots.* PhD thesis, Carnegie-Mellon University, 1989.

[6] Robert Guzikowski. Control computing for autonomous mobile systems. Master's thesis, Carnegie Mellon University, 1984.

[7] Karl Kluge and Charles E. Thorpe. Explicit models for robot road following. In Charles E. Thorpe, editor, *Vision and Navigation: The Carnegie Mellon Navlab*, chapter 3. Kluwer Academic Publishers, 1990.

[8] Motorola, 2900 South Diablo Way, Tempe, AZ 85282. *MVME133XT VMEmodule 32 Bit Monoboard Microcomputer User's Manual*, 1988.

[9] Motorola, 2900 South Diablo Way, Tempe, AZ 85282. *MVME3330 Ethernet Controller User's Manual*, 1988.

[10] Motorola, 2900 South Diablo Way, Tempe, AZ 85282. *MVME340A Parallel interface/timer User's Manual*, 1988.

[11] Dean A. Pomerleau. Neural network based autonomous navigation. In Charles E. Thorpe, editor, *Vision and Navigation: The Carnegie Mellon Navlab*, chapter 5. Kluwer Academic Publishers, 1990.

[12] Dong Hun Shin. *High Performance tracking of explicit paths by roadworthy mobile robots.* PhD thesis, Carnegie-Mellon University, 1990.

[13] *VxWorks User's Manual.* 1351 Ocean Ave, Emeryville, CA 94608, 1989.

[14] R. Wallace, A. Stentz, C. Thorpe, H. Moravec, W. Whittaker, and T. Kanade. First results in robot road-following. In *Proc. IJCAI-85*, August 1985.